Math

UIUCDCS-R-72-538

COST EFFECTIVE ANALYSIS OF NETWORK COMPUTERS

BY

WILLIAM J. BARR

August 1972

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-72-538

COST EFFECTIVE ANALYSIS OF NETWORK COMPUTERS

BY

WILLIAM J. BARR

August 1972

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

*This work was supported in part by Contract No. NSF GJ 28289 and was was submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science, August 1972.

ABSTRACT

With the advent of network computers, a new area of computer systems analysis has evolved. Unfortunately, most of the work to date merely extends the previously existing theory of communications. Our analysis is predicated on the assumption that a geographically distributed network computer is quite different from telephone networks and individual computing centers. The potential advantages, from a businessman's point of view, of network computers are discussed and some of the more important problems which arise are presented. We take a look at several existing networks and examine their goals and solutions to these kinds of problems. We develop a priority assignment technique for the individual centers that comprise the network and determine load leveling rules for the entire network. In addition, we analyze this system using classical queueing theory techniques.

ACKNOWLEDGEMENT

The author would like to express his most sincere gratitude to his advisor, Professor E. K. Bowdon, Sr., for his advice and criticisms. Without his help and encouragement, this paper would not exist.

Thanks also goes to my wife, Rhoda, for her aid in proofreading and encouraging me to write in the English language.

Special thanks go to Phyllis Sloat for her beautiful typing.

TABLE OF CONTENTS

Page

1. INTRODUCTION.....1

2. THE ECONOMIES AND ECONOMICS OF NETWORK COMPUTERS.....6

2.1. Money--The Root of all Research.....6

2.2. The ARPA Network.....13

2.3. The TUCC Network.....18

2.4. The Distributed Computer System.....20

2.5. The ALOHA System and the Problem of Communication Media.....25

3. COST EFFECTIVE PRIORITY ASSIGNMENT.....30

3.1. A Scenario of Cost Effectiveness.....30

3.2. Priority Assignment.....35

3.3. Load Leveling in a Network of Centers.....45

3.4. Summary.....48

4. QUEUEING THEORY ANALYSIS OF THE SINGLE SERVER PRIORITY CASE WITH BRIBING AND RENEGING.....52

5. CONCLUSION.....63

LIST OF REFERENCES.....65

1. INTRODUCTION

Communication systems have long been recognized as being of great importance to all facets of human society. Dating from Greek mythology and Pheidippides' historic run, praises have been heaped upon those who succeeded in delivering important messages before the messages became unimportant. Paul Revere was immortalized for his success in this field as were the riders of the Pony Express. However, all of these legendary and romantic figures were residents of an earlier age when information could be important even when several days old. The invention of the telegraph, the telephone, and, later, the radio changed the nature of the communications industry. The excitement of crawling through the enemy camp with critical information which had been carefully memorized (and the paper it was written on eaten) has been replaced by the thrill of watching a maze of wires and lights spit forth the unscrambled transmission from the guy down the road. The fastest runner with the swiftest horse has been replaced by majestic telephone lines as the dominant hero of the industry. Seconds have replaced days as the critical time units in which transmissions must occur and ulcers have replaced sore muscles as the physical characteristics of the hard working messenger. The communication industry has grown up!

By the middle of the twentieth century the phone company had grown to be perhaps the single most important component of industry. The United States had become a nation filled with telephones and all of them were interconnected. In short, the phone company had built a very large network. During the 1960's most of the components of the network became electronic. It was no longer necessary to argue with an operator to place a long distance call, it was sufficient, and cheaper, to simply dial the proper sequence of numbers and it was possible to reach almost anyone in the country.

The 60's were also the glory years for the electronic digital computer. The computer left the day to day drudgery of number crunching behind--usually as a background job--and moved into the business world. Information storage, retrieval, and processing became the primary occupation of the computer. To the frustration of many an information systems manager, the information he wanted typically took days to arrive at the computing center but only fractions of a second to process. Thus the information which was available to management, though available at the push of a button, was usually days old. The stage was now set for the merger of the two entities, the telephone network and the computer, into one which we call the NETWORK COMPUTER.*

Two definitions are required before we proceed. We define a computing center to be either a single processor or a geographically local and independent group of interconnected processors. We now define a network computer to be any interconnected group of computing centers. Thus a multiprocessor computing center is also a network computer (frequently called a subnetwork). The apparent ambiguity in this definition is important since the statements we shall make in this paper will apply to all multicenter network computers without regard to whether the centers are comprised of a single processor or a subnetwork. Furthermore, we note that there is no requirement that the interconnections be provided by the phone company. We specifically wish to allow microwave, radio, and light to be possible media for communication between centers.

---

*We use the term network computer as opposed to computer network for many reasons, the most immediate of which is to avoid confusion with the logic networks which are used in computers and are frequently called computer networks. Furthermore, the analysis of electrical networks by computers is well known as "computer network analysis."

Since the initial conception of the idea, many network computers have been designed and some have reached the working stage. Until recently, these have all been constructed primarily as experiments--investigations into the unknown. With support from several organizations--primarily the Advanced Research Projects Agency (ARPA) and the National Science Foundation (NSF)--the goal of these networks has been to advance technology to make it possible to build network computers which can aid society or at least make some money. These experiments have succeeded in advancing technology, but they have also succeeded in opening a veritable Pandora's box of problems, both economic and political in nature, which appear to be much more difficult to solve than the remaining technological problems. We propose to concern ourselves here with some of the miriad of problems which are economic in nature. Our primary motivation for this choice is that computers tend to be very expensive guinea pigs and unless techniques can be found to solve these problems, the political problems may become irrelevent. The continued existence of network computers is predicated upon the ability of designers and managers to show them to be economically viable.

Network computers offer one significant advantage that is not offered by single processor installations: the availability of large quantities of resources at a cost which is significantly less than would be required if each center purchased these resources individually. These resources are basically of two types: unique resources and common resources. Unique resources are those resources which, for whatever reason, can be made available only at a small number of locations. Examples of unique resources would include the array processing capabilities of the ILLIAC IV or the CDC STAR or specially developed software systems and proprietary programs. Common resources

are those resources which are available at many centers. Increasing the common resources of a center has two distinct advantages. First of all, the computing services become a more reliable commodity. If a processor goes down in a single processor installation, computing services are lost for the duration of the down time. However, if the installation is part of a network computer, some of its tasks can be sent to other installations within the network for processing with a minimum loss of service. The second advantage to increasing the amount of common resources is that the network achieves load leveling capabilities. By load leveling, we mean the capability to transmit tasks between centers for the purpose of improving the overall throughput of the network. What load leveling does, from a user's point of view, is improve the stability of the computing services which he uses since the peak demand periods at his home center can be smoothed into the slack periods at another center.

These advantages, while enticing, remain advantages only if they can by achieved in a cost effective manner. In this paper we shall direct our analysis towards developing cost effective tools for use in network computers.

In Chapter II we briefly discuss some of the problems which arise in network computers from making available increased amounts of each type of resource through a network. We primarily take the businessman's point of view in the analysis with the goal of developing concepts to make network computers attractive to the people who have money to spend in exchange for services. (For insight into these problems we are deeply grateful to Einar Stefferud of Einar Stefferud and Associates, Santa Monica, California.) In this chapter we also examine several networks, both existing and planned, to see how the businessman's problems have affected or might affect policies

and operations. Additionally, we take a look at the communications problem and see what types of communications media are most cost effective in different types of environments.

In Chapter III we develop priority assignment and scheduling algorithms designed to simplify load leveling in a network computer. The system which evolves from this algorithm is analyzed using queueing theory techniques in Chapter IV.

Finally, in Chapter V, we take time to get a perspective of the results obtained in this paper. We also take a brief look at the road ahead for network computers and indicate some of the roadblocks which remain.

2. THE ECONOMIES AND ECONOMICS OF NETWORK COMPUTERS

"'The time has come,' the Walrus said, 'to talk of many things'..."
Of research done, and networks built, and money making things.

With apologies to Lewis Carroll

In this chaper we are concerned with defining and discussing considerations which must be taken into account in order for network computers to become viable commodities in the computer marketplace. The potential advantages, from a businessman's point of view, of network computers are discussed and some of the more important business problems which arise are presented. Finally, we take a look at several existing networks and examine their goals and solutions to these kinds of problems.

2.1 Money--the Root of all Research

Until recently little justification was needed for the construction and maintenance of network computers. Each project was viewed as a novel and noble investigation into the unknown. During such investigations, inefficiencies could be tolerated and practicality certainly was not an immediate goal. Computers, however, tend to be very expensive guinea pigs and the continued existence of network computers is predicated upon the ability of designers and managers to demonstrate their economic viability.

With the technological problems virtually solved, it is becoming apparent that no matter whose point of view one takes, the only economically justifiable motivation for building a network computer is resource sharing. However, the businessmen, the users, the people with money to spend, could not care less whose resources they are using for their computer runs. They

care only that they receive the best possible service at the lowest possible price. The computing center manager who cannot fill this order will soon find himself out of customers. Let us place ourselves in the shoes of a computing center manager, attempt to define some of his problems, and then see how a network computer might help to solve them.

"The best possible service..." is, in itself, a tall order to fill. The computing center manager finds himself in a position to offer basically two kinds of computing services: contract services and demand services. Contract services are those services which the manager agrees to furnish, at a predetermined price, within specified time periods. Examples of this type of service include payroll runs, billings, and inventory updates. Each of these is run periodically and the value placed by the businessman on the timely completion of the task is large. Demand services are those services which, though defined in advance, may be required at any time and at a possibly unknown price. Frequently the only previous agreements made refer to the type of service to be delivered and limits on how much will be demanded. Examples of tasks which are run on a demand basis include research, information requests, and program debugging runs. University computing centers generally find that most of the services which they offer are of this type.

Though the classification of computing tasks into one of these pidgeon holes may sometimes be difficult, the distinction is important because of the differing nature of the problems which arise in the successful delivery of services. We will make no attempt to classify computing tasks as belonging to either class but the division is convenient in terms of problem definition. Each type of service generates its own set of problems. For clarity we will discuss them separately and then return to tie them together.

The major problem which arises when a computing center offers contract services is how to be certain that any guarantees made can be kept. Computer down time is an unfortunate but all too real fact of life. Ensuring that equipment does not go down during critical production periods is one of the most difficult tasks in the computing business. Murphy's law (a well known heuristic which states, among other things, that if anything can possibly go wrong, it will, and at the worst possible time) warns us to beware of putting all of our eggs in one basket. Every computing center manager who offers contract services should have a solid and acceptable answer to the critical question, "What do I do when my computer breaks down?" If he wishes to ensure that he can meet all commitments the only answer is to transfer contract tasks to another processor. This is where network computers enter the picture. If the center is part of a network computer, tasks can easily and quickly be transferred to another center (presumably one with resources compatable with the center in question) for processing. The user is happy because his work got done, the manager of the recipient center is happy because he got extra work, and the manager of the stricken center is happy (or at least relieved) because he was able to meet his commitments.

Associated with the advantage of the manager's being able to offer a more reliable guarantee is the increased expansion capabilities associated with being part of a network computer. Should a center become so busy that its existing computing capabilities are strained, he would clearly like to expand. However, if he has only a single processor, me may find it difficult to continue expanding until another, or larger, processor is working. If he is part of a network computer it would be possible for him to use someone else's resources to process his overflow work until his

new resources are available. Furthermore, if the increased work is not sufficient to justify expansion of his own facilities, it is possible for him to act as a broker for resources at other centers.

The broker concept is also a valid one when talking about the availability of unique resources. The existance, for example, of a CDC STAR on the network offers the manager extensive array processing capabilities which he can provide to his customers with little additional effort on his own part. Similar services, of less dramatic impact, can likewise be provided through the network. The user is happy because he can get this increased service through his own broker (again with little additional effort) and the manager of the center with the unique resources is happy for the increased business. The broker concept has been widely discussed in the literature and details will not be included here (see especially [29] and [40]). (As an aside, we feel that network computers provide the only feasible means for such super computers to become viable. They are so fast and so powerful that only an extremely large customer base will suffice to keep them in business. Only through a network computer can this base be economically provided and maintained.)

Offering computing services on a contract basis can be an ulcer-producing job. However, with the potential for stability which the network computer offers, the problem of guarantees is somewhat alleviated. It should be noted that the word is "alleviated"--not eliminated. Network computers do not completely satisfy Murphy's law. It is possible for the communications equipment, like all other machines, to break down. However, the probability that both the processor and the communications equipment will be down at the same time is considerably less than that for the processor alone. Thus we feel that network computers can be of significant service in the business of offering contract computing services.

The second type of service which is offered by computing centers is called demand service. The expectations of the user of demand services are less clearly defined than those of the user of contract services. The traditional viewpoint associated with this type of user is that he simply wishes to get his tasks processed as quickly as possible. Discrimination is frowned upon and, apart from partitioning tasks by size, a first come-first served policy is generally used for resource allocation. We feel that this is an outdated policy. The basic assumption upon which it is predicated is that all tasks are of equal value to the user and hence should be treated as equal by the computing system. We do not doubt that this assumption was valid, even a few years ago, when no task was given to the computer unless it was reasonably important. This, however, is no longer the case. Computing centers handle a wide variety of tasks ranging from urgent to trivial. Thus all tasks should not be treated equally. If the user can somehow specify the value of a task, the computing system should recognize this value and treat the task accordingly. In Chapter 3 we shall develop a priority assignment system which allows the user to specify not only the value of a task but also a deadline which represents the latest moment at which the task has that value. We feel that this system comes much closer to serving the needs of the user who requests demand services. He has more control over the fate of his tasks and the system respects his requests and accords the privileges due them.

Given the system sketched above, the job of the task scheduler becomes much more difficult. The scheduler now has the additional task of keeping track of and meeting deadlines. This need not be as difficult as it seems. If the scheduler can have enough foresight to know when tasks are in danger of missing a deadline, it will be able to send some of these tasks to other centers who can meet the deadline. Again, network computers

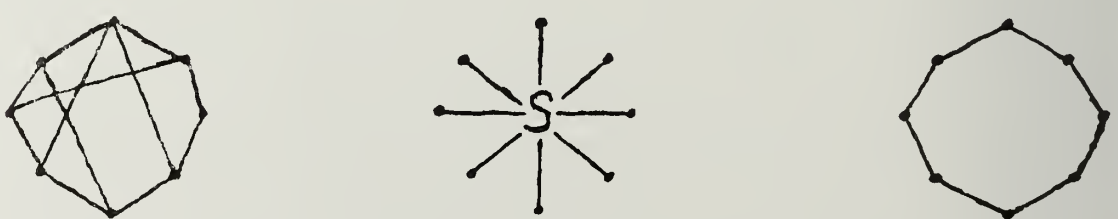
are a tremendous help in solving the problem. This idea is similar to that presented in the discussion of contract services. However, it is much more dynamic in nature and requires more careful supervision. A system which develops from attending to these concerns is generally called a load leveling system. Our version of a load leveling system is described in Chapter 3.

We have seen that a network computer can offer many advantages to managers of computing centers. However, building a network computer is not cheap. The problem now facing the industry is how to make these advantages available at a price we can afford. This is again the problem of cost effectiveness. Different networks take different approaches in presenting a cost effective facade to the businessman. Whether or not this facade will withstand the pressures of adversity will decide the fate of the networks.

There are still problems which remain unsolved. Especially sticky are those problems which arise in a network made up of competitive centers. The manager of a computing center is not pleased by the prospect of sending business to a competitor. Yet this is exactly the method we have proposed. Billing is another problem which arises in this open market situation. How does the manager of the recipient center determine the price to charge for services rendered? These are very difficult management problems. One feasible solution is to construct networks made up of centers working for the common good. Another is to have one center which is impartial--an overflow center--which does not compete for outside business. This approach is probably impractical, although a policy of mutually exclusive clientele may be feasible.

In the following sections we shall examine the approaches taken by several networks. These networks were selected for study because we feel they are representative of the wide variety of philosophies and ideas which enter into the design of network computers. Though our prejudices may be reflected, we do not intend to imply that ony one network is better than any other.

There are three basic multiprocessor (or network) topologies. Type 1, the distributed network (Figure 2.1a), is unique because it offers more than one path between any two centers. This topology was chosen for the ARPA network which is discussed in section 2.2. Type 2, the star network (Figure 2.1b), is distinguished by a central message switch. All center to center communication must pass through this switch. The TUCC network (section 2.3) utilizes a modified version of the star network. In the third type, the ring structure (Figure 2.1c), all communication is along the same path and there is only one path between any two centers. This topology is being investigated for the Distributed Computer System (section 2.4). As we shall see, each of these topologies has some advantages and disadvantages and each can be cost effective in some setting.



a) A Distributed Network

b) A Star Network

c) Ring Structure

Figure 2.1 Possible Network Topologies

2.2 The ARPA Network

We begin our survey of network computers with the ARPA (Advanced Research Projects Agency) network. We do this for several reasons. It is the most widely discussed network computer (see especially [15], [17], [24], [28], [31], and [38]) and has probably provided the single most important impetus in the advance of network computer technology. It is also the largest fully automatic network computer in operation (with the possible exception of defense networks about which little information is available) both in terms of geographic distribution and number of centers. (We do not include Cybernet since human intervention is generally required for its successful operation.) The initial topology for the ARPA network is shown in Figure 2.1 (diagram taken from [38]).

The ARPA network has provided most of the current widely used terminology in the field, some of which will be used in the discussions that follow. The most widely used term is "node", the name given to a center and its communications computer. The center has been dubbed the "Host" (initially only one processor though there is no necessary restriction on tthe number of processors in a host) while the communications computer has been named "IMP" for Interface Message Processor.

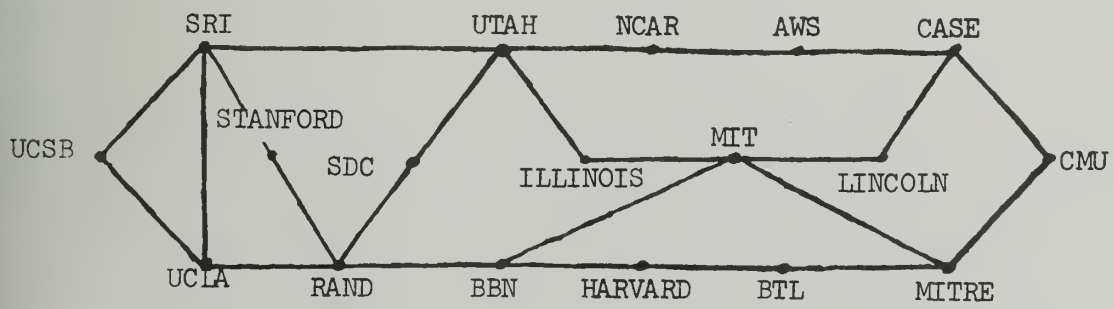


Figure 2.2 The ARPA Network

The design of the ARPA network was influenced by two major considerations. Since this was the first widely funded fully computer run network, a primary goal was to explore network computer technology. This work has progressed well enough for us to remove it from further consideration. A second, and more relevent, goal was to interconnect those research centers across the country to which ARPA contributed substantial financing. This interconnection would allow the sharing of unique resources and, hopefully, eliminate some duplication of effort.

One of the primary elements of cost in any network computer is topology. The topology of the ARPA network as shown in Figure 2.2 can be called distributed. It is not a fully interconnected distributed network since this would imply direct paths between any two nodes. The most important consideration which led to the choice of a distributed network was reliability The central switch network is unreliable since there is only one path between any two nodes--through the central switch. The central switch should not be dismissed lightly (see section 2.3 on the TUCC network) but it does not fulfil the needs of the ARPA network. The consideration of reliability also leads to difficulty in the ring network. However, the ring structure has been modified for the Distributed Computer System (see section 2.5) and the reliability of this type of network has been improved. The geographical distribution of the ARPA network and its traffic considerations rule out the use of a ring structure.

Once it had been decided that the ARPA network was to be a distributed network, much effort was spent in optimizing the final topology. This work has been well referenced in the literature (see especially [24]) and will not be discussed here.

One of the most important side effects which resulted from the choice of a distributed network was the sophistication which is required in the IMP. Due to technological considerations, a distributed network which is not fully interconnected must use store and forward message switching. This requires considerable error checking and decision making capabilities in the IMP. For these reasons the ARPA network communications computer is significantly more expensive than that required for some other networks. Available figures indicate that the basic IMP will run in the neighborhood of $40,000 to $50,000 [44]. However, the IMP by itself is not of much value, and from the beginning, work has continued on developing an intelligent terminal which can be connected to a Host. The initial work in this area resulted in a Terminal Information Processor (or TIP) which included the HP 316 IMP in its cost of about $92,000. However, the TIP has proved unsatisfactory for some applications and further work has resulted in the ARPA Network Terminal System (ANTS) which includes the HP 316 and a PDP 11. Work is now progressing to connect the $120,000 ANTS system to a CDC 6600 at Ft. Belvoire. If successful, this work would make the entire ARPA network to look like simply an intelligent terminal (obviously one with large resources) to each Host [44].

We have seen that membership in the ARPA network will require a minimum initial investment of at least $90,000. In addition, $20,000 to $35,000 in annual maintenance and ARPA priviledges can be anticipated [44]. What then are the motivations which justify the high cost of membership in ARPA? To the research oriented nodes the primary motivation is obviously

the capability of using a wide variety of unique resources. Anyone doing work which requires large amounts of matrix arithmetic and array processing would clearly prefer to have the ILLIAC IV (in a working version) available than any of the currently popular general purpose processors. For many of the major universities, access to the ILLIAC IV alone may justify the expense of ARPA network membership. Another valuable unique resource is the University of Utah interactive graphics system. This likewise provides a significant amount of justification for membership. As was previously mentioned, a network computer is a virtual necessity for these specialized and very expensive unique resources and we have just seen that these resources can be very important to the network.

However useful the availability of these unique resources may be, we feel that, for the ARPA network to become a viable commodity without being subsidized, a wider variety of advantages must be offered. As ARPA becomes more commercial, we feel that we shall see significant use of the common resources of the network. Since communication costs are expected to be low (about $.30 per megabit after the first 4000 megabits) it would be quite feasible to shop around for the node with the quickest turn-around time before submitting a task or, perhaps more interestingly, do some load leveling on a network-wide basis. If this is the case, any computing center with either unused or excess capacity could become a candidate for membership in the ARPA network. Assuming it is possible to solve the political problems which would obviously arise in such an association (definitely a non-trivial assumption), this approach could be the key to cost effective operation of the ARPA network.

Perhaps the most radical possibility opened by the ARPA network is that its successful implementation could lead to the elimination of most medium and large scale machines. The ANTS system clearly has the potential to operate as a remote job entry station for all kinds of tasks--including interactive procedures. Given this capability and assuming that the ARPA network traffic capacity is great enough, we can envision a time when, instead of purchasing a medium or large scale computing system, the most economical action would be to purchase an ANTS type system and use already existing resources. This approach would probably require the use of brokers and would undoubtedly lead to the development of more specialized machines which could do certain types of tasks more cost effectively than general purpose machines. One factor that will hinder the development of this type of system is the well known "blame it on the computer" syndrome. When errors are made, top management does not like to blame things on a machine. They instead would like to have someone close by who is responsible for the success and failure of the computing system. It is this tendency which leads those responsible to want their own machines. If a person is responsible for a product he would at least like to have control over the development of the product. Far from demeaning this tendency, we encourage it. This reluctance is well justified and its existence will help to encourage responsible development of computing systems. The idea is to solve as many problems as possible before they arise. Responsible development, along with open ears, are the keys to developing cost effective computing systems.

2.3 The TUCC Network

The TUCC (Triangle Universities Computation Center) network is a system which is entirely different from the ARPA network. In the first place it is contained entirely within the state of North Carolina, thus serving a much smaller area than the nationally distributed ARPA network. Just as significant, it is currently operational and working in a cost effective manner, while ARPA is still getting the bugs out. For these reasons, and others, it is an interesting network to examine.

The TUCC network is a regional homogeneous network consisting of an IBM 370/165 (at Research Triangle Park), and IBM 360/75 (at the University of North Carolina/Chapel Hill), and two IBM 360/40's (at Duke University/ Durham and North Carolina State University at Raleigh). (Also connected to the 370/165 are several secondary terminals at state educational institutions but we will not concern ourselves with these in this discussion.) These four processors are connected in a central switch topology as shown in Figure 2.3. This topology was chosen for primarily one reason: Communication from the three universities is expected to be mainly to Research Triangle Park, and direct university to university communication rare.[37] Thus extra links would be economically unjustifiable. Total cost to the universities for belonging to the TUCC network is in the range of $13,000 to $14,000 per

Figure 2.3 The TUCC Network

month [43]. Though this may seem high at first glance, one should realize that this fee includes approximately 25% of the computing capacity of the 370/165. Adding this amount of capability to any of the centers individually would cost around $19,000 per month [43]. Thus, if the capacity can be used, membership in the TUCC network is certainly economically justified.

The primary motivation in developing the TUCC network was economics. Thus far, the performance of the network seems to justify most of the grossly general statements which have been made earlier in this paper. The 370/165 is primarily available to provide remote job entry and interactive services to the three universities. It should be noted that most of the tasks processed at Research Triangle Park represent excess capacity demanded by users at the three universities. That this demand is significant can be seen from the fact that it was necessary to install an IBM 370 since the 360/75 which it replaced in September, 1971 was frequently saturated. This clearly supports the argument which we made earlier for increasing the availability of common resources.

The increase in the availability and utilization of unique resources has not been nearly as dramatic. A few proprietary programs exist and are utilized. However, the existence of these does not, alone, justify the cost of the network. Two fairly unique advantages were obtained, however, with the formation of the network. First, the universities became better able to attract faculty and staff who require large scale computing services for research and teaching. As any educator can tell you, this attraction can be an important help to the relevent university departments. A second important consideration was that TUCC was able to attract outstanding systems programmers which the individual universities could neither afford

nor keep busy. Attracting the top men in the field always helps a university--or a network of universities.

Though currently operating in a cost effective manner, the TUCC network is far from becoming sedentary. Plans are currently under way to enable the 360's at the various universities to handle overflow from the 370. Obtaining this capability is especially important in a university environment where vacations, and hence due dates for term projects, can be staggered. Although other services are provided by the TUCC network perhaps deserving more attention in this discussion) the load sharing capabilities which are made available are sufficient to provide significant monetary savings, as well as improvements in service, to the three universities involved.

We saw earlier that the ARPA network was designed to share unique resources and that there was at least some justification for feeling that these alone could support a network computer. We have just seen that it is possible to maintain a network computer built to share common resources. One wonders how a network computer might fare if both types of resources were made available.

2.4 The Distributed Computer System

We have already looked at the ARPA network, which is nationally distributed, and the regional TUCC network. The logical third type of network to discuss is a local network. The Distributed Computer System (DCS) fills this bill with a great many ingenious ideas which deserve discussion. In addition, DCS, is an example of the third type of network topology, the ring network.

The Distributed Computer System is an experimental computer network which is being designed and built at the University of California at Irvine under the direction of David J. Farber. When completed, DCS is expected to provide medium and small scale computing services to the users at UCI. Providing complete computing services is never an easy thing to do. To make the development of DCS even more interesting, a number of restrictive design objectives were imposed on the system. The following were the design objectives proposed for DCS [21]:

1) Reliability: The system should be invulnerable to total failure. It is a fact of life that individual compoents will fail, but the failure of any individual component should have minimal effect on the operation of the rest of the system. This capability is called "soft fail".

2) Low initial cost: Like most institutions, UCI is short of money. The initial hardware cost of the system was restricted to less than $250,000. This effectively ruled out the medium scale systems used by most universities of this size, such as IBM's 360/50, DEC's PDP-10, or XDS's Sigma 7.

3) Incremental expansion ability: The system should be capable of expansion of virtually any type, at a relatively low expansion cost.

4) Variety of language systems: Since this is to be a university computing system, a wide variety of computer languages and resources should be available.

5) Modest system programming requirements: It would be nice to keep the system as simple as possible.

We now examine the ways in which these design objectives affected the final system design. The most important objective is that the system be capable of soft fail. In a single processor system, if that processor goes down the whole system is down. This violates the soft fail objective, so we are forced to consider multiprocessor possibilities.

As we have seen before, there are three basic multiprocessor (or network) topologies. All others degenerate to one of these. Type one, the distributed network, solves this problem since between any two processors in the net there exist at least two possible communication paths. (This is not true in general, but if it weren't true for our system we wouldn't consider it at all.) Another property, however, makes this configuration undesirable for DCS. The interface between the processor and the net must be capable of store-and-forward message switching and intelligent enough to perform all message routing functions, which might be quite complex. As is usually the case, this increased interface sophistication leads to increased cost. It turns out that if we wished to start with a modest network of five or six processors, the interfaces alone would account for at least half, if not all, of the allotted initial funding. Type two, a network with central message switch, has the same problem as the single processor system. If the switch dies, all interprocessor communication dies, too.

By using type 3, the ring structure, we can at least avoid these problems. In the maintenance of the communications ring, two things can go wrong: the ring itself can fail or some interface between the ring and a processor can fail. About the only way the ring can fail is if it is broken. Although the probability of a cable being broken is near zero, we can guard against this failure by placing secondary alternatives to the primary ring path which can be invoked by a hardware mechanism. One method

considered to accomplish this objective was to force the ring to have an even number of ring interfaces (nodes) by inserting an extra dummy node if necessary, and connecting node N to node N+2, node N+1 to node N+3, etc., as well as to their immediate neighbors. This "daisy chain" scheme has the drawback of making it more difficult to include new nodes in the network. Early literature indicated that a simpler alternative, buying two rings initially and installing both at the same time was feasible [21]. With synchronization problems taken care of, it might also be possible to use both at the same time when both are operational, but at worst we are left with a complete backup system for the ring cable. More recently, however, the designers seem to be seriously investigating the daisy chain scheme [26].

The other possible problem arises if one of the nodes goes down and interferes with messages in a roadblock fashion. This problem can be avoided by building a hardwired "short-circuit" device which can be engaged by the node itself or by an outside source. This short-circuit device logically disconnects the processor from the ring. In addition, the ring interface is constructed in such a way that store-and-forward message switching techniques are required only between a processor and its own ring interface. This allows a substantial decrease in transmission time of messages and simplifies the ring interface. The result is that the node can be built far more inexpensively than any other type of communications processor (perhaps $500) [21]. The initial planned structure of DCS is shown in figure 2.4.

So much for the objectives of reliability and cost. The other objectives can also be easily met with a ring structure. Incremental expansion presents no problem. Expansion of the system requires only hooking up an additional ring interface and attaching it to the new processor. There is

no theoretical limit to the expansion of the system, although in practice the number of processors the ring can efficiently handle is limited to a number currently unknown. Expansion is economical because of the low cost of the ring interface. A variety of language systems can be obtained by attaching small language-oriented processors to the ring. It is well known that this approach can be very cost-effective. The system programming objective may be violated by building a distributed operating system but we adopt this approach anyway to maintain the soft fail capability of the ring.

We note that the critical design objectives for DCS are reliability and low initial cost. We have seen that a ring structure is the most feasible way to meet these objectives, and that the others can also be met using this approach.

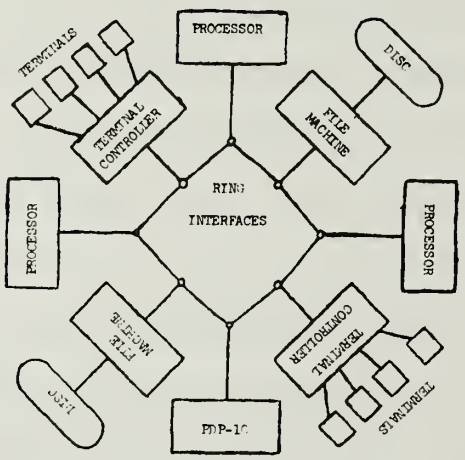


Figure 2.4. The Distributed Computer System

This relatively detailed discussion was provided in order to give a feeling for the real goals of the system and some of the problems encountered. Other important aspects of the system, especially the distributed operating system, are deserving of discussion. To do so, however, would be to deviate from the purpose of this paper. The reader is referred to references [21] and [22] for more complete discussions.

We now have a pretty good idea of the cost of DCS. How effectively it will operate remains an unanswered question. It is, after all, an experimental network which is not even operational at this time. It is important to note that speed was not a major design objective. Some of the decisions that were made to ensure reliability were undoubtedly made at the cost of reduced speed. The degree to which this tradeoff will affect the system is yet undetermined.

We are encouraged by the development of DCS. If all of the remaining hurdles can be negotiated, we feel that the flexibility provided in the design and the services which can be offered by the system are a real bargain for the price.

2.5 The ALOHA System and the Problem of Communication Media

In each of the networks studied so far, the communication media used has been wire. In the continental United States, where telephone networks already exist, or in local networks, wire is currently the most cost effective communications media and is well documented in the literature [17, 21, 23, 32, 38, 43]. That this situation exists is not, however, due to any inherent qualities in the wire but is due primarily to convenience. One proposed system which has selected an alternate medium is the ALOHA System. As we shall see, of the many feasible media, wire is perhaps the least cost effective choice for the ALOHA System.

The ALOHA System is being built by the University of Hawaii and is intended to provide computing services to users at the University's main campus in the Manoa Valley on Oahu, a four-year college at Hilo, Hawaii, two-year community colleges on the islands of Oahu, Kauai, Maui, and Hawaii, and associated research institutes within a 200-mile radius of the

University. Long-range plans include extending the communications system throughout the Pacific area, possibly even for ship to shore use. Some fast, inexpensive method is needed to connect an IBM 360/65 computer at the University's main campus, called the Kahuna, with remote access terminals (Keikis) at all of the above-mentioned locations [1,2].

It is expected that ALOHA will eventually have a very large number of Keikis, but that relatively few of them will be in use at any given time. The expected usage pattern of a single active Keiki is that a user will type in some message to be sent to the Kahuna and await a response. It is important to note that the action of typing a message takes a few seconds, and it takes the user at least a second or so to realize he has received a response and react to it, but the Kahuna's results are available in milliseconds. This motivates the requirement that the communications process should take a maximum of a few seconds, but it also means that a minimum communication time of as much as half a second is not unreasonable from the user's point of view.

The usual method of remote access communication is by wire, most often over leased telephone lines. This approach can be cost-effective when the distances involved are not too great, the wires already exist, or they would be in use much of the time. It is not appropriate for ALOHA because new wires would be needed to some of the islands (existing wires in many places provide only low-quality voice-grade communication and are, therefore, not sufficiently reliable for data transmission), many Keikis are located at relatively large distances from the Kahuna, and each Keiki is expected to be used only a few hours a day. To make matters worse, extending ALOHA throughout the Pacific would make the cost of wire communication prohibitive because of the distances involved, and its use by ships at sea would be quite impossible.

Among the alternatives to wire communication are radio, light waves, and an amphibious marine troop transporting reels of magnetic tapes. The latter must be rejected immediately because it is too expensive and too slow. At present ALOHA is being built using radio communication, but research on the feasibility of light wave communication using light emitting diodes is also under way. Radio communication was chosen because it is inexpensive to use over large distances (in fact, radio messages could be beamed in by satellite from all over the world), the Keikis can be mobile, and a communications protocol can be established which takes advantage of the burst nature of messages from a single Keiki while ignoring all the inactive Keikis. It is this protocol which makes ALOHA so interesting.

While the Kahuna is kept busy with computing tasks, an HP2115A named Menehune (after a legendary Hawaiin elf) serves as its link to the radio communication channel. It is Menehune's job to multiplex messages between Kahuna and a large, indeterminate number of users. Menehune uses two 100KHZ channels, one for receiving messages from the Keikis, and the other for sending messages to the Keikis. The complete makeup of the ALOHA System is shown in Figure 2.5.

Multiplexing messages from Menehune to the Keikis is relatively easy. For traffic in this direction there is only one sender and messages can be queued and transmitted sequentially. Transmission from the Keikis to Menehune is quite another story, however, and the reader is enthusiastically referred to the literature for a complete discussion [1,2]. We will concern ourselves, in this paper, only with the results of the investigation.

Because of the anticipated usage pattern of the Keikis, the designers of the ALOHA System have chosen to allow communication in burst mode only. Under this constraint, Menehune can be said to have a capacity of about 24,000 bits per second. In practice, since message arrivals are virtually

random, the capacity will be much less. Studies by the author agree, within reason, with the results of Bortels [8] and the theoretical results [1,2] that for message packets of about 700 bits, the system can support between 140 and 160 active users if the transmission rate is about two packets per

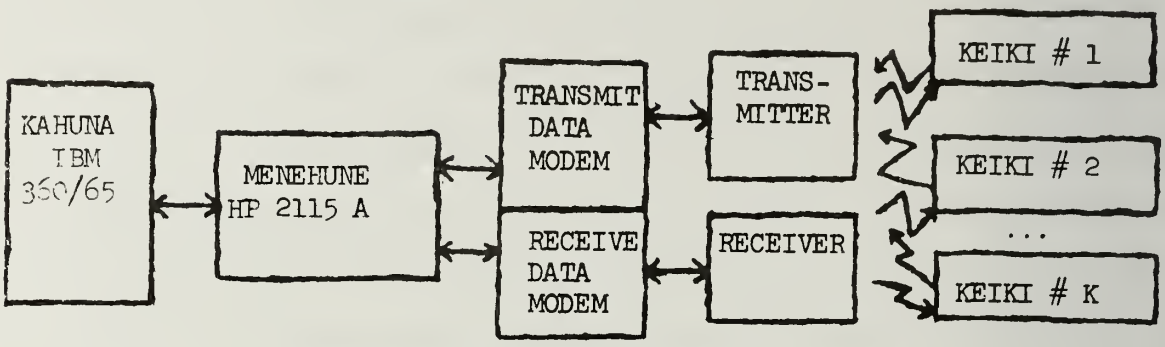


Figure 2.5 The ALØHA System

Keiki per minute. (Bortels actually indicates that this number may be a bit high but we will stick by these numbers). This is obviously a significant number of users especially when one realizes that they are all using the same channel.

The investment needed to become part of the ALOHA system is anticipated to be one teletype and the $3,000 Keiki (a one-time cost) [1]. We feel that this is a very reasonable amount since it allows the user access to the complete computing capabilities of the University of Hawaii. And, should the user keep the Keiki long enough, the average monthly cost will be low enough to allow almost anyone, regardless of location, to join the system.

According to the definitions presented earlier, the ALOHA System is not really a network computer. The Keikis, after all, support only terminals, not processors. The potential remains, however, to utilize the methods proposed for the ALOHA System in computer to computer communication. Obviously, the number of computers which could be connected in this way is less than the number of Keikis which can be connected because of the nature

of the traffic, but the potential remains. Perhaps radio communication would not be feasible in the more densely populated areas of the world because of interference. However, in at least one instance, the ALOHA System, radio has been shown to be a cost effective alternative to wire for communication in network computers.

3. COST EFFECTIVE PRIORITY ASSIGNMENT

3.1 A Scenario of Cost Effectiveness

Until recently, efforts to measure computer efficiency have centered on the measurement of resource (including processor) idle time. A major problem with this philosophy is that it assumes that all tasks are of roughly equal value to the user and hence the operation of the system.

As an alternative to the methods used in the past, we propose a priority assignment technique designed to represent the worth of tasks in the system. We present the hypothesis that tasks requiring equivalent use of resources are not necessarily of equivalent worth to the user with respect to time. We would allow the option for the user to specify a "deadline" after which the value of his task would decrease, at a rate which he can specify, to a system determined minimum. With this in mind, we propose a measure of cost effectiveness with which we can evaluate the performance of a network with an arbitrary number of interconnected systems, as well as each system individually.

We define our measure of cost effectiveness γ, as follows:

$$\gamma = (Lq/M)^{\alpha(1-(M-Lq)/\sum_{i=o}^{R-1} \beta(i))}$$

where

L_q is the number of tasks in the queue,

M is the maximum length queue,

R is the number of priority classes,

α is a measure (system-determined) of the "dedicatedness" of the CPU to the processing of tasks in the queue,

and $$\beta(i) = (R-i) \sum_{j=1}^{n} (g(j)/f(j))$$

where g(j) is the reward for completing task j (a user specified function of time),

and f(j) is the cost (system determined) to complete task j.

The term $(Lq/M)^{\alpha}$ is a measure of the relevance of the queue to processing activities. Similarly, we can look at $\beta(i)$ as a measure of resource utilization. Note that $\beta(i)$ indicates a ratio of reward to cost for a given priority class and is sensitive to the needs of the user and the requirements imposed on the installation. It is user sensitive because the user specifies the reward and is installation sensitive because the cost for processing a task is determined by the system. The measure of CPU dedicatedness (α), on the other hand, is an entirely installation sensitive parameter.

The first problem which becomes apparent is that which arises if $\sum_{i=o}^{R-1} \beta(i) = 0$. This occurs only in the situation where there is exactly one priority class (i.e., the non-priority case). We will finesse away this problem by defining

$$(M-Lq)/\sum_{i=0}^{R-1} \beta(i) = 0$$

for this case. Intuitively, this is obvious, since the smaller this term gets, the more efficiently (in terms of reward) a system is using its resources. Furthermore, in the absence of priorities, the order in which tasks are executed is fixed, so this term becomes irrelevant to our measure of cost effectiveness. Thus, for the non-priority case, we have

$$\gamma = (Lq/M)^{\alpha}$$

which is simply the measure of the relevance of the queue to processing activities. This is precisely what we want if we are going to consider only load-leveling in non-priority systems. However, we are interested in the more general case in which we can assign priorities.

An estimate of the cost to complete task j, f(j) is readily determined from the user-supplied parameters requesting resources. Frequently these estimated parameters are used as upper limits in resource allocation and the operating system will not allow the program to exceed them. As a result, the estimates tend to be high. On the other hand, lower priorities are usually assigned to tasks requiring a large amount of resources. So the net effect is that the user's parameters reflect his best estimate and we may be reasonably confident that they truly reflect his needs.

At the University of Illinois computing center, for example, as of July 26, 1971, program charges are estimated by the following formula:

$$\text{cents} = a(X + Y)(bZ + c) + d$$

where X = CPU time in centiseconds,

Y = number of I/Ø requests,

Z = core size in kilobytes,

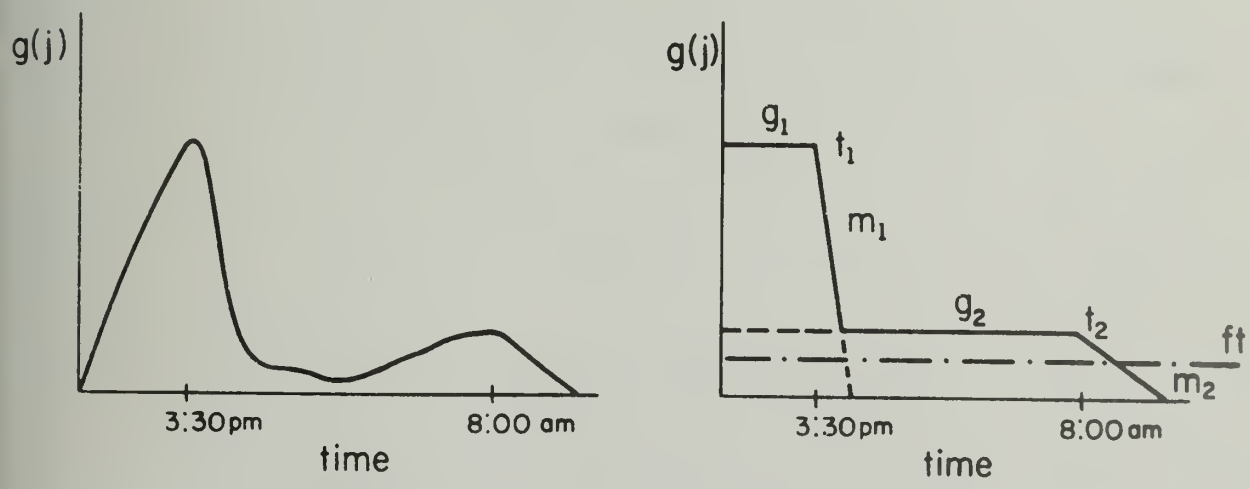
a, b, c are weighting factors currently having the values 0.04, 0.0045, and 0.5, respectively.

and d is an extra charge factor including $1.00 cover charge plus any special resources used (tape/disk storage, card read, cards punched, plotter, etc.).

The main significance of the reward function g(j) specified by the user is that it allows us to determine a deadline or deadlines for the task.

Typically we might expect g(j) to be a polynomial in t, where t is the time in the system. For example, the following thoughts might run thorugh the user's head: "Let's see, its 10:00 a.m. now and I don't really need immediate results since I have other things to do. However, I do need the output before the 4:00 p.m. meeting. Therefore, I will make 3:30 p.m. a primary deadline. If it isn't done before the meeting, I can't use the results before tomorrow morning, so I will make 8:00 a.m. a secondary deadline. If it isn't done by then I can't use the results, so after 8:00 a.m. I don't care."

The function g(j) this user is thinking about would probably look something like Figure 3.1a. Now, this type of function poses a problem in that it is difficult for the user to specify accurately and would require an appreciable amount of overhead to remember and compute. Notice, however, that even if turnaround time is immediate, the profit oriented installation manager would put the completed task on a shelf (presumably an inexpensive storage device) and not give it to the user until just before the deadline--



(a) Ideal Function (b) Approximate Function

Figure 3.1 Example of a User's Reward Function

thus collecting the maximum reward. As a result, there is little reason to specifying anything more than the deadlines, the rewards associated with meeting the deadlines, and the rate of decrease of the reward between deadlines, if any. Applying this reasoning to Figure 3.1a we obtain Figure 3.1b. Note that this function is completely specified with only six parameters (deadlines t_1, t_2; rewards g_1, g_2; and rates of decrease m_1, m_2).

In general, we may assume that g(j) is a monotonically non-increasing, piecewise linear, reward function consisting of n distinct sets of deadlines, rewards, and rates of decrease. Thus we can simple specify g(j) with 3n parameters where n is the number of deadlines specified.

Note that, in effect, the user specifies an abort time when the g(j) he specifies becomes less than f(j). If the installation happens to provide a "lower cost" service, $\tilde{f}(j)$ and if $g(j) > \tilde{f}(j)$, this task would be processed, but only when all the tasks with higher g(j) had been processed.

Now, what we are really interested in, is not so much an absolute reward, but a ratio of reward to cost. Since f(j) is, at best, only an estimate of cost, we cannot reasonably require a user to specify an absolute reward. A more equitable arrangement would be to specify the rewards in terms of a ratio g(j)/f(j) associated with each deadline. This ratio is more indicative of the relative worth of a task, both to the system and to the user, since it indicates the return on an investment.

3.2 Priority Assignment

Let us now turn our attention to the development of a priority assignment scheme which utilizes the reward/cost ratios described in the previous section. We egin by quantizing the continium of reward/cost ratios into R distinct intervals. Each of these intervals is then assigned to one of R priority classes 0, 1, 2, ..., R-1 with priority 0 being reserved for tasks with highest reward/cost ratios and priority R-1 for tasks with reward/cost ratios of unity of less. A task entering the system will be assigned a priority according to its associated reward/cost ratio.

We want to guarantee, if possible, that all priority 0 tasks will meet their deadlines. Furthermore, if all priority 0 tasks can meet their deadlines, we want to guarantee, if possible, that all priority 1 tasks will meet their deadlines and, in general, if all priority k tasks can meet their deadlines, we want to guarantee that as many priority class k+1 tasks as possible will meet their deadlines. To facilitate the priority assignment, we introduce the following notation:

For priority k, let T_i denote the i^{th} task. Then we assume for each T_i that we receive the following information vector:

$$(T_i, g/f, d_i, \tau_i, s_i)$$

where

T_i is an identifier,

g/f is the reward/cost ratio associated with meeting the task's deadline,

d_i is the task's deadline associated with g/f,

τ_i is the maximum processing time for the task,

and $s_i = d_i - \tau_i$, is the latest time at which the task may start processing and still be assured of meeting its deadline.

Now since each task has an associated deadline and maximum processing time, we can us the resulting latest start time as the basis for assigning positions to tasks within a priority class. A last come, first served rule will be uused to break ties. Additionally, we will use a compacting scheme to ensure that as many tasks as possible start processing before their latest start times. More formally our algorithm may be stated as follows:

Priority Assignment Algorithm

First, we assign a new task a priority based on its g/f ratio, say priority k. Then within class k, its position is determined as follows:

1. Beginning with the last priority k task (that is, the task with latest deadline) search forward until two tasks, T_{j-1} and T_j, are found such that $d_{j-1} < d_i \leq d_j$. Insert T_i between the two tasks, assign it a start time $s_i = d_i - \tau_i$, and renumber the tasks behind T_j accordingly (i.e., T_i becomes T_j, T_j becomes T_{j+1}, etc.).
2. Now, if $s_{j-1} + \tau_{j-1} \leq s_j \leq s_{j+1} - \tau_j$ there is sufficient float time between T_{j-1} and T_{j+1} for T_j to be processed on time and the priority assignment is complete. However, if either $s_{j-1} + \tau_{j-1} > s_j$ or $s_j + \tau_j > s_{j+1}$, a deadline might be missed; so we proceed with Step 3.
3. Compacting Scheme. Let f_j denote the float time between any two tasks T_{j-1} and T_{jj}, where f_j is defined:

$$f_j = s_j - (s_{j-1} + \tau_{j-1})$$

Then, F_j, the total float time preceding T_j, is given by:

$$F_j = \sum_{k=1}^{j} f_j = s_j - \tau + \sum_{k=1}^{j} \tau_j.$$

where τ is the current time.

Now, starting with task T_j, if $s_j + \tau_j > s_{j+1}$ and $F_j \geq s_j + \tau_j - s_{j+1}$ we assign a new starting time to T_j given by:

$$s_j = s_{j+1} - \tau_j.$$

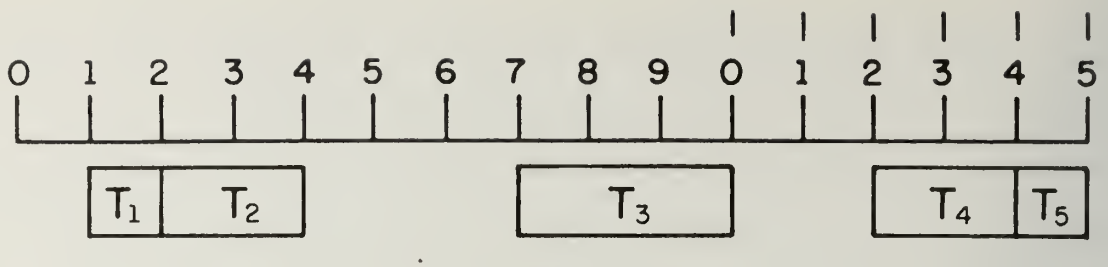
and we continue with T_{j-1}, T_{j-2}, etc. until we encounter a task T_k, $k \neq j$, such that $s_k \leq s_{k+1} - \tau_k$. (Note that T_{j+1} and all its predecessors are guaranteed to meet their deadlines.)

4. However, if $s_j + \tau_j > s_{j+1}$ but $F_j < s_j + \tau_j - s_{j+1}$, we do not assign a new start time to T_j. Instead we leave the start time at its latest critical value, even though it may not start processing at that time. We observe that many tasks may not require all of the processing time specified by their maxima, and as a result sufficient float time may be created later to enable the task, T_j, to meet its deadline.

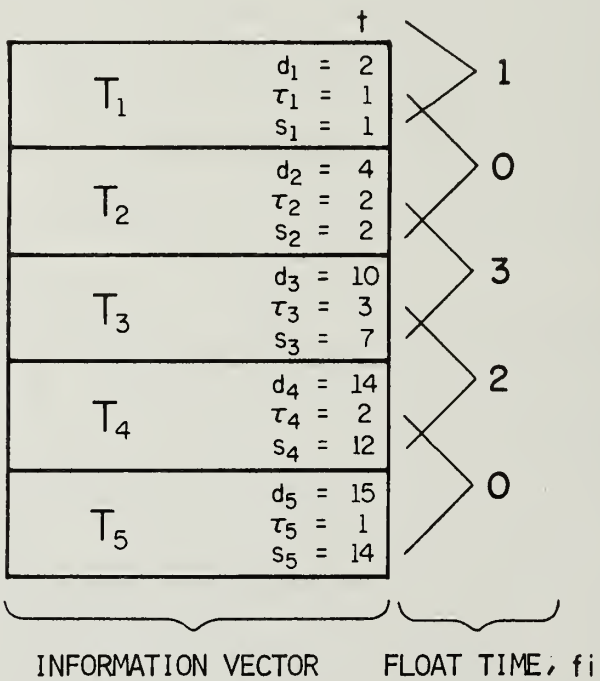
Examples.

Several examples will now be given to illustrate the efficacy of the algorithm. Suppose we have determined that a task, T_i, should have priority k and that at time t = 0, the state of priority class k is that shown in Figure 3.2. (Note that since all priority class k tasks have similar g/f, we need not show these ratios.) Notice that forming the float time column is analogous to forming a forward difference table. In each of the following examples we assume that Figure 3.2 is the initial state of priority class k.

i) Suppose the information vector (with g/f omitted) for T_i is $(T_i,6,1,5)$. Beginning with T_5, we observe that $d_2 < d_i \leq d_3 < d_4 < d_5$. So we insert T_i between T_2 and T_3 and renumber the tasks accordingly.



A) SCHEDULE OF TASKS.



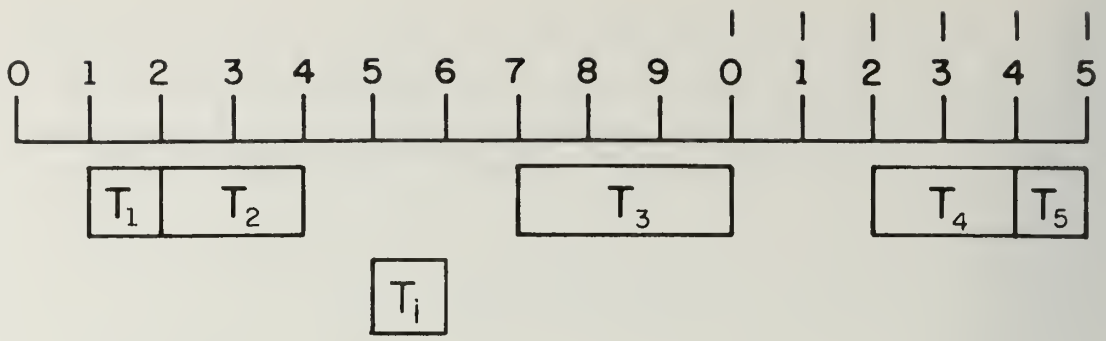
B) INFORMATION VECTORS.

FIGURE 3.2 – STATE OF PRIORITY CLASS K AT TIME $t = 0$

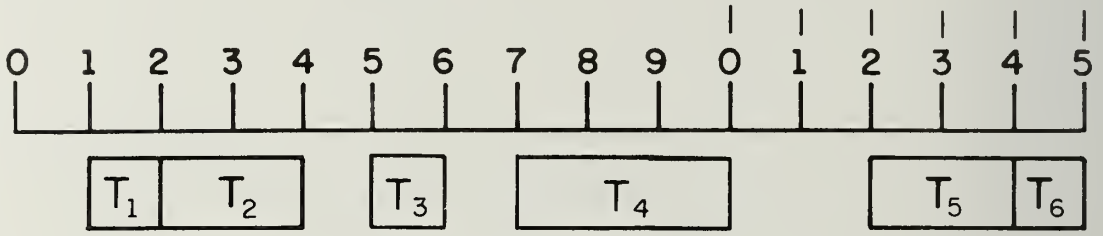
Now $s_2 + \tau_2 \leq s_3 \leq s_4 - \tau_3$ since $2 + 2 \leq 5 \leq 7 - 1$, so the priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 3.3.

ii) Suppose instead that the information vector for T_i is $(T_i,9,2,7)$. We find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $7 + 2 > 7$ and a deadline could be missed. But $F_3 \geq s_3 + \tau_3 - s_4$ since $4 \geq 2$, so we assign a new start time to T_3: $s_3 = s_4 - \tau_3 = 7 - 2 = 5$. Now $s_2 \leq s_3 - \tau_2$ since $2 \leq 5 - 2$, so the priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 3.4. (Note the effect of the last in first out rule for breaking ties on start times.)

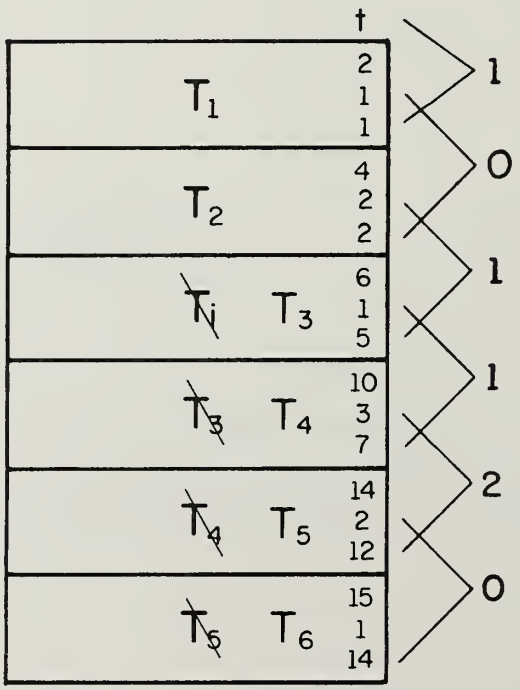
iii) Next suppose the information vector for T_i is $(T_i,9,4,5)$. We find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $5 + 4 > 7$, and a deadline could be missed. But $F_3 \geq s_3 + \tau_3 - s_4$ since $4 \geq 4$ so we assign a new start time to T_3: $s_3 = s_4 - \tau_3 = 7 - 4 = 3$. Next $s_2 + \tau_2 > s_3$ since $2 + 2 > 3$, so we assign a new start time to T_2: $s_2 = s_3 - \tau_2 = 3 - 2 = 1$. Now $s_1 + \tau_1 > s_2$ since $1 + 1 > 1$, so we assign a new start time to T_1: $s_1 = s_2 - \tau_1 = 0$. The priority assignment is complete and all tasks are guaranteed to meet their deadlines. The resulting state of priority class k is shown in Figure 3.5.



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.

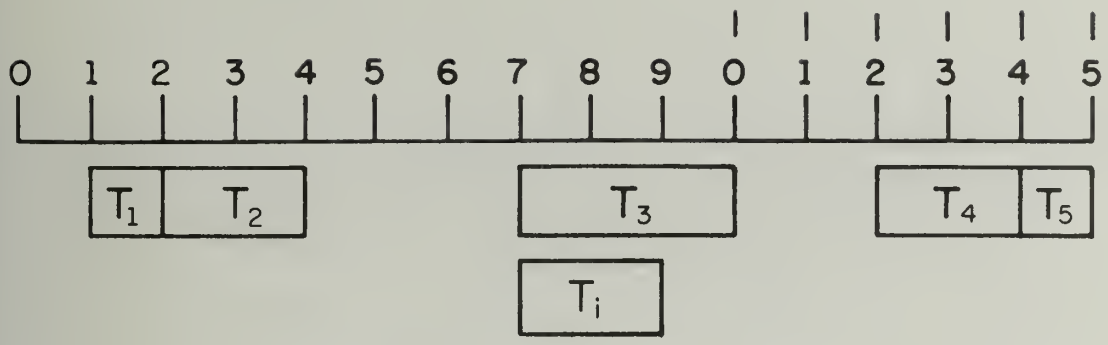


B) SCHEDULE OF TASKS AFTER ASSIGNMENT.

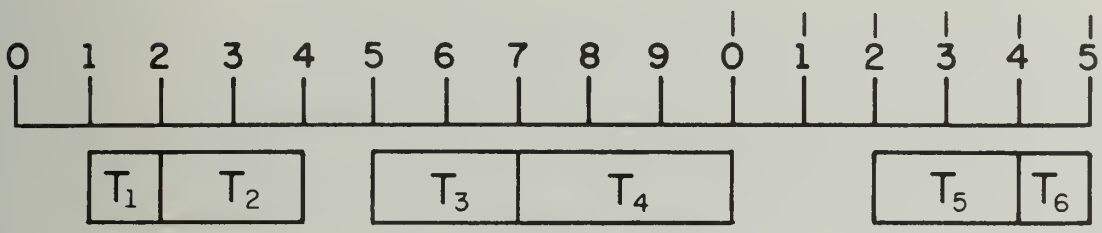


C) INFORMATION VECTORS AFTER ASSIGNMENT.

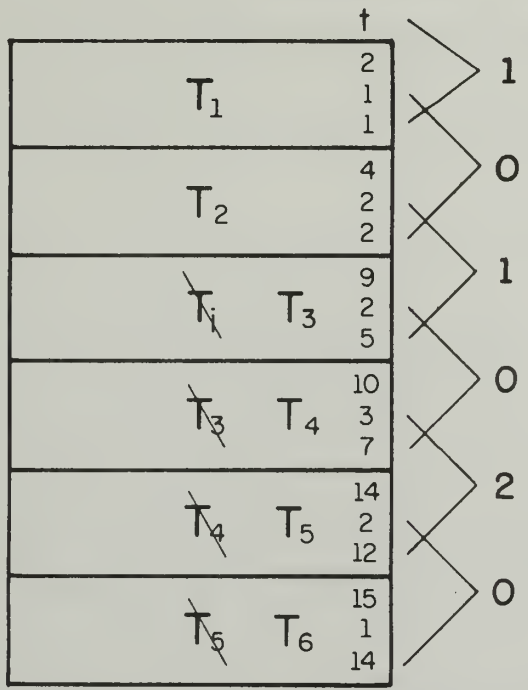
FIG. 3.3 - RESULTS OF PRIORITY ASSIGNMENTS (i)



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.

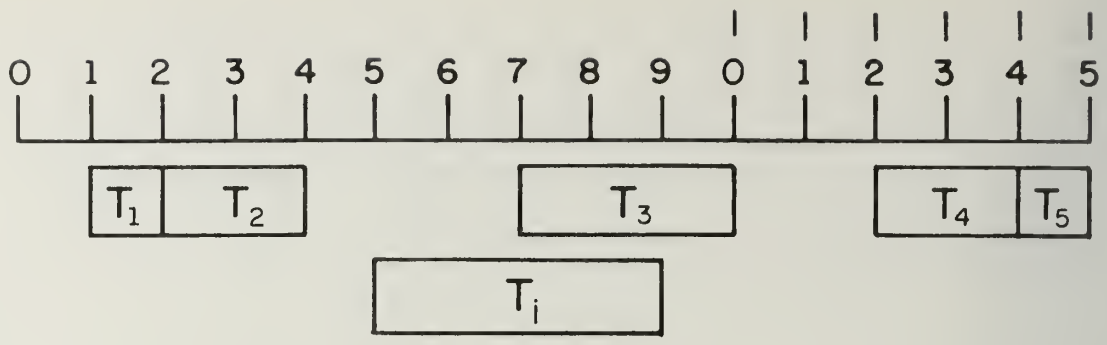


B) SCHEDULE OF TASKS AFTER ASSIGNMENT.

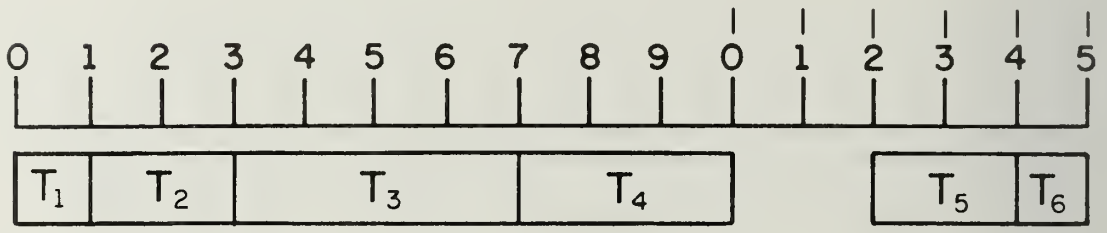


C) INFORMATION VECTORS AFTER ASSIGNMENT.

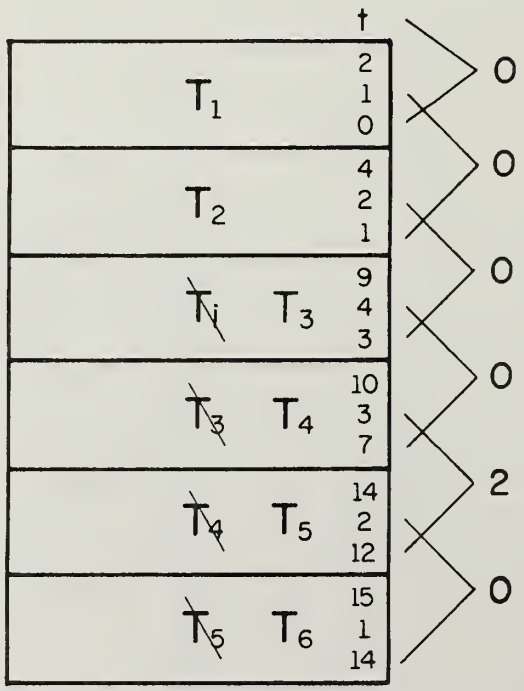
FIG. 3.4 - RESULTS OF PRIORITY ASSIGNMENTS (ii)



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.



B) SCHEDULE OF TASKS AFTER ASSIGNMENT.



C) INFORMATION VECTORS AFTER ASSIGNMENT.

FIG. 3.5 - RESULTS OF PRIORITY ASSIGNMENTS (iii)

iv) As a final example, suppose that the information vector for T_i is $(T_i,9,5,4)$. As before, we find that $d_2 < d_i \leq d_3 < d_4 < d_5$, so we insert T_i between T_2 and T_3 and renumber the tasks accordingly. However, $s_3 + \tau_3 > s_4$ since $4 + 5 > 7$, and a deadline could be missed. Furthermore, $F_3 < s_3 + \tau_3 - s_4$ since $1 < 2$, and the compacting scheme will not help us. Instead we leave the start times at their latest critical values and hope that sufficient float time is created later to enable the tasks to meet their deadlines. The results of this assignment are shown in Figure 3.6. Note that T_4 is the task which is in danger of missing its deadline.

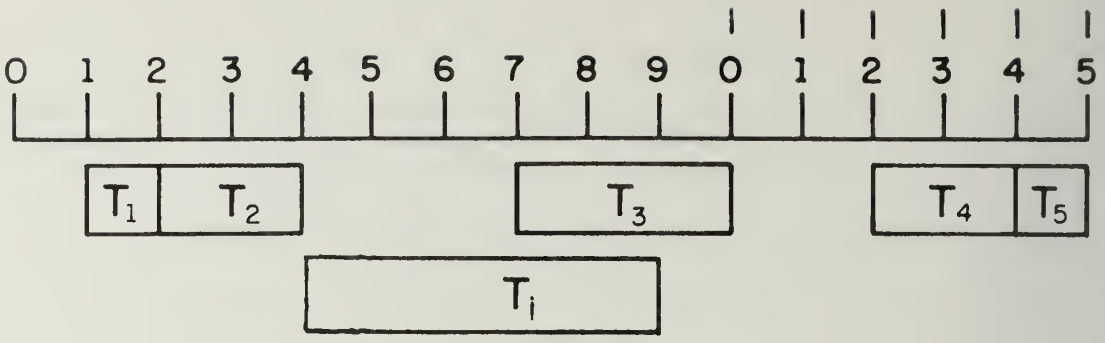
The last example brings up the problem of what to do with a task whose deadline is missed. We simply treat it as though it had just entered the system using the next specified deadline as the current deadline. If no further deadlines are specified, the task is assigned priority R-1 and will be processed accordingly.

When a processor finishes executing a task the following scheduling algorithm is used to determine which task is to be processed next. Generally, the algorithm takes the highest priority task in the queue that is closest to its latest starting time.

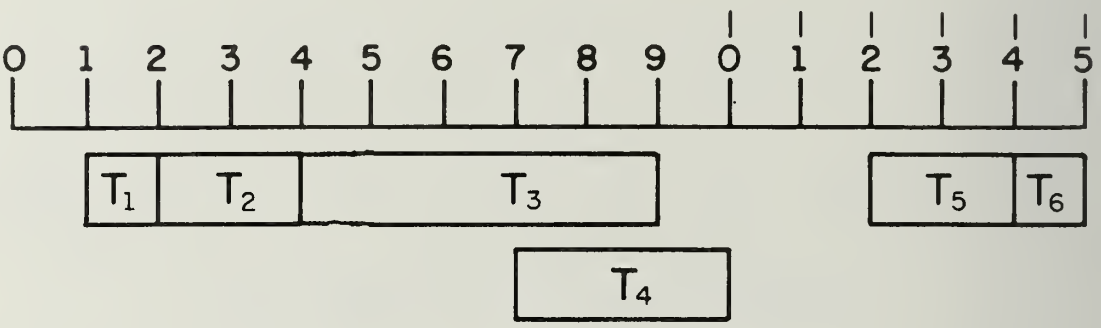
Scheduling Algorithm

Beginning with k=0, and using ℓ as an index,

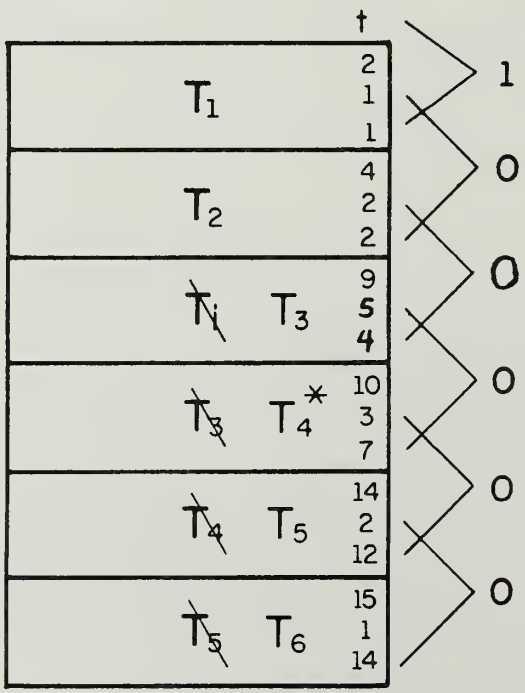
1. We examine the float time, f_1, for the first task in priority class k. Then for ℓ = k+1:
2. If f_1 of priority class k $\leq \tau_1$ for the first task of priority class ℓ, we set k=ℓ and continue with Step 1. Otherwise we continue with Step 3.



A) SCHEDULE OF TASKS BEFORE ASSIGNMENT.



B) SCHEDULE OF TASKS AFTER ASSIGNMENT.



C) INFORMATION VECTORS AFTER ASSIGNMENT.

FIG. 3.6 - RESULTS OF PRIORITY ASSIGNMENTS (iv)

3. Set $\ell = \ell+1$ and continue with Step 2 until all priority classes have been considered. Then continue with Step 4.
4. Assign the first task, T_1, in priority k to the available processor.

The effect of this scheduling algorithm is quite simple. It instructs the scheduler to schedule the important tasks first and then, if there is sufficient time, schedule those lower priority tasks in such a way that as many deadlines as possible are met.

In the foregoing we have tacitly assumed that each task enters the system sufficiently before its deadline to allow processing. The two algorithms taken together facilitate meeting the deadlines, where possible, of the higher priority tasks. Those tasks which do not meet their deadlines will tend to be uniformly late.

3.3 Load Leveling in a Network of Centers

Thus far we have been concerned only with cost effectiveness in a single center. Next, let us consider the more general problem of load leveling within a network of centers. Each center may contain a single computer or a subnet of computers. The topological and physical properties of such networks have been illustrated [1,23,32,42] and will not be discussed here.

We wish to determine a strategy which optimizes the value of work performed by the network computer. That is, to guarantee that every task in each center will be processed, if possible, before its deadline and only those tasks that offer the least reward to the network will miss their deadlines. Implicit in this discussion is the simplifying assumption that any task can be performed in any center. This assumption is not as restrictive as it may sound since we can, for the purposes of load leveling,

partition a nonhomogeneous network into sets of homogeneous subnetworks which can be considered independently. Thus, in the discussion which follows we will assume that the network computer is homogeneous.

We define the measure of cost effectiveness for a network of N centers, γ_N, as follows:

$$\gamma_N = \sum_{i=1}^{N} \omega_i \gamma_i \quad \text{given that} \quad \sum_{i=1}^{N} \omega_i = 1$$

where

the ω_i are weighting factors that reflect the relative contribution of the i^{th} center to the overall computational capability of the network,

and γ_i is the measure of cost effectiveness for the i^{th} center. Note that if a center is a subnet of computers, we could employ this definition to determine the measure of cost effectiveness for the subnet. We also let c_{ij} denote the cost of communication between centers i and j; and t_{ij} the transmission time between centers i and j.

Ideally, we want the network computer to operate so that all tasks within the network are processed before their deadlines. If a task is in danger of missing its deadline, we want to consider it as a candidate for transmission to another center for processing. The determination of which tasks should be transferred follows the priority assignment (i.e., priority 0 tasks in danger of missing deadlines should be the first to be considered, priority 1 tasks next, etc.).

We note that this scheme may not discover all tasks that are in danger of missing their deadlines. In order to discover all tasks that might be in danger of missing their deadlines, we would require a look ahead scheme to determine the available float time and to fit lower priority

tasks into this float time. The value of such a scheme is questionable, however, since we assume some float time is created during processing and additional float time may be created by sending high priority tasks to other centers. Also, the overhead associated with executing the look ahead scheme would further reduce the probable gain of such a scheme.

The determination of which center should be the recipient of a transmitted task can be determined from the measure of cost effectiveness of each center. Recall that the measure indicates the worth of the work to be processed within a center. Thus, a center with a task in danger of missing its deadline will generally have a larger measure than a center with available float time. Thus, by examining the measures for each center, we can determine the likely recipient of tasks to be transmitted. These centers can in turn, examine their own queues and bid for additional work on the basis of their available float time. This approach has a decided economic advantage over broadcasting the availability of work throughout the network and transmitting the tasks to the first center to respond. The latter approach has previously been discussed and disguarded by Farber [21].

Once a recipient center has been determined, we would transmit a given task only if the loss in reward associated with not meeting its deadline is greater than c_{ij}, the cost of transmitting the task between centers and transmitting back the results.

When a task is transmitted to a new center its deadline is diminished by t_{ij}, the time to transmit back the results, thus ensuring the task will reach its destination before its true deadline. Similarly, the reward associated with meeting the task's deadline is diminished by c_{ij}, since

this represents a reduction in profit. Then the task's g/f ratio is used to determine a new priority and the task is treated like one originating in that center.

This heuristic algorithm provides the desired results that within each center all deadlines are met, if possible, and if any task is in danger of missing its deadline, it is considered for possible transmission to another center which can meet the deadline.

3.4 Summary

We have introduced a priority assignment technique which, together with the scheduling algorithm, provides a new approach to resource allocation. The most important innovation in this approach is that it allows a computing installation to maximize reward for the use of resources while allowing the user to specify deadlines for his results. The demand by users upon the resources of a computing installation is translated into rewards for the center.

This approach offers advantages to the user and to the computing installation. The user can exercise control over the processing of his task by specifying its reward/cost ratio which, in turn, determines the importance the installation attaches to his requests. The increased flexibility to the user in specifying rewards for meeting deadlines yields increased reward to the center. Thus the computing installation becomes cost effective, since for a given interval of time, the installation can process those tasks which return the maximum reward. A notable point here, is that this system readily lends itself to measurement.

The measure of cost effectiveness is designed to reflect the status of a center using the priority assignment technique. From its definition, the value of the measure depends not only on the presence of tasks in the system but upon the priority of these tasks. Thus the measure reflects the worth of the tasks awaiting execution rather than just the number of tasks. Therefore, the measure can be used both, statically, to record the operation of a center and, dynamically, to determine the probability of available float time. This attribute enables us to predict the worth of the work to be performed in any center in the network and facilitates load-leveling between centers.

We have spent a good deal of time discussing what this system does and the problems it attempts to solve. In the interest of fair play, we feel that we should also discuss the things it does not do and the problems it does not solve.

In Chapter 2, one of the proposed benefits of a network computer was that it was possible to provide, well in advance, a guarantee that, at a predetermined price, a deadline would be met. This guarantee is especially important for scheduled production runs, such as payroll runs, which must be processed within specified time periods. The system as presented does not allow for such a long range guarantee. To implement such an option, however, is not difficult. Notice first of all that we say nothing that requires rewards to be strictly monetary in nature. This allows us to say that in order to meet such absolute guarantees we simply modify the reward to include the loss of goodwill which would be incurred should such a deadline be missed. Perhaps the easiest way to implement this would be to reserve priority class zero for tasks whose deadlines were previously guaranteed. Under this system we could assure the user, with confidence, that the deadlines could be met at a predetermined price. The difficulty

which arises under such a modification is that we have no way of ensuring that the user does not submit tasks to priority class zero which have not been guaranteed in order to get the best possible service at a (possibly) lower price. This problem is one of security and remains unsolved in this paper.

A second problem with the system is that the algorithms presented in no way attempt to optimize the mix of tasks which would be processed concurrently in a multiprogramming environment. This is a valid difficulty which we can answer only by saying (perhaps hopefully) that the system will tend to keep itself balanced. A common strategy in obtaining a good mix is to choose tasks in such a way that most of the tasks being processed at one time are input/output bound (this is especially common in large systems which can support a large number--five or more--tasks concurrently). Generally smaller tasks are the ones selected to fill this bill. Under our system, it seems reasonable that the higher priority classes will tend to contain the smaller and less expensive tasks since priority is assigned on the basis of a cost multiplier which is user supplied. It seems fair to conclude that a user would be much more reluctant to double (give a reward/cost ratio of 2) the cost of a $100 task than to double the cost of a $5 task. This reluctance, we feel, will tend to keep a good mix present in a multiprogramming environment.

The final loose end which we would like to tie down here is the problem of what to do with a task if (horror of horrors) all of its deadlines are missed. There are basically two options, both feasible for certain situations, which will be discussed. We feel that the ultimate decision as to which is best belongs to the folks who manage the centers so we will attempt to present the alternatives objectively without any intent to influence their decision.

The first alternative made obvious by the presentation is that when a task misses all of its deadlines the results it would produce are of no further use. Continued attempts to process a task in this instance would be analogous to slamming on the brakes after your car hits a brick wall; simply a waste of resources. Thus we might say that if the deadlines are firm, a task which misses all of its deadlines should be considered lost to the system.

On the other hand, it is conceivable that the results produced by a task are of value even after the last deadline is missed. In this case it is reasonable to require that a system offer a "low cost" service under which tasks are processed at a reduced rate but at the system's leisure. A danger in this system is that it is conceivable that, if run without outside supervision, the system could become saturated with low cost tasks to the detriment of more immediately valuable work. This actually happened at the University of Illinois during early attempts to institute a low cost option. From the results of this experience, it is safe to say that it will never happen more than once to the same installation manager. The confusion and headaches which resulted from the saturation were more than enough to institute protective measures.

Despite a few admitted shortcomings, we feel that our system represents a definite positive step in the analysis of network computers.

Our approach treats a network computer as the economic entity that it should be: a market place in which vendors compete for customers and in which users contend for scarce resources. The development of this approach is a first step in the long road to achieving economic viability in network computers.

4. QUEUEING THEORY ANALYSIS OF THE SINGLE SERVER PRIORITY CASE WITH BRIBING AND RENEGING

The priority assignment and scheduling algorithms presented in Chapter 3 give rise to an interesting queueing theory problem. We are presented with a situation in which customers offer overt bribes (a term introduced in queueing theory by Kleinrock [33]) to improve their queue position. Furthermore these customers can, and will, renege (give up their queue position and leave the system) if their bribe is not accepted and their demand for services not met. It is to the analysis of this problem that we direct our efforts in this chapter.

In the analysis we will assume the following:

1. Input population: We assume an infinite input population with average arrival rate for priority class r of λ_r. The arrival rate for the system is

$$\lambda = \sum_{r=0}^{R-1} \lambda_r$$

where R is the number of priority classes.

2. Queue discipline: R priority classes. Finite queue with maximum length M > 1. Bribes are random with arbitrary cumulative distribution function B. Priority class for an entering task is assigned on the basis of its bribe.
3. Service mechanism: Single server with constant service rate μ.
4. Service pattern: Within a priority class on the basis of closeness to deadline. Tasks renege and are lost to the system if their deadline is missed. Tasks from priority class r are scheduled for processing on the basis of available float time from classes r-1 and higher as described in Chapter 3.

5. Reneging mechanism: Reneges for priority class r are distributed in a Poisson manner about mean rate α_r. The renege rate for the system is distributed about mean

$$\alpha = \sum_{r=0}^{R-1} \alpha_r .$$

We find the steady state probabilities for the system in the following manner. First look at the no priority case (R = 1). Let $P_n(t)$ be the probability that at time t an arrival finds (n-1) waiting tasks already in the system; that is, he is n^{th} away from service. Then $P_0(t)$ is the probability that an arriving task goes directly into service. Using classical techniques (well discussed in the literature [11, 18, 32, 39]) we proceed by generating difference equations. We find

$$P_0(t+\Delta t) = P_0(t)(1-\lambda\Delta t)(1) + P_1(t)(1-\lambda\Delta t)(\mu\Delta t) + o(\Delta t)^2 .$$

Continuing in the usual way we find that

$$P_0'(t) = -\lambda P_0(t) + \mu P_1(t)$$

which leads to the steady state equation

$$P_1 = (\lambda/\mu)P_0 .$$

To find the steady state equations for $M \geq n > 1$ we adopt the approach taken by Ancker and Gafarian [5]. Since any of the (n-1) customers in the queue can renege during a time interval of length Δt, the probability that one of these will renege is $(n-1)\alpha\Delta t$. Furthermore, because of the ordering of tasks within a priority class, we know that should a task renege, it is the task which is first in the queue. Utilizing this result we find that

$$P_n(t+\Delta t) = P_n(t)(1-\lambda\Delta t)(1-\mu\Delta t)(1-(n-1)\alpha\Delta t)$$
$$+ P_{n-1}(t)(\lambda\Delta t)(1-\mu\Delta t)(1-(n-2)\alpha\Delta t)$$
$$+ P_{n+1}(t)(1-\lambda\Delta t)(\mu\Delta t)(1-n\alpha\Delta t)$$
$$+ P_{n+1}(t)(1-\lambda\Delta t)(1-\mu\Delta t)(n\alpha\Delta t) + o(\Delta t)^2.$$

Continuing again we find that

$$P_{n+1} = \left(\frac{\lambda+\mu+(n-1)\alpha}{\mu+n\alpha}\right)P_n - \frac{\lambda}{\mu+n\alpha} P_{n-1}.$$

By induction we can show that

$$P_n = \frac{\lambda^n}{\prod_{i=0}^{n-1}(\mu+i\alpha)} P_0 \qquad n = 1,2,\ldots,M+1$$

and

$$P_0^{-1} = \sum_{n=0}^{M} (\lambda^n) / \prod_{i=0}^{n-1}(\mu+i\alpha).$$

Note that P_{M+1} is the steady state probability that an arriving task finds the queue full.

The no priority case which we have just solved has been discussed previously in the literature. Though different in form, our results agree with those of Ancker and Gafarian [5,6]. The problem which has not been solved is that of a single server multiple priority class system. This is the system described in Chapter 3. The interesting thing about the problem is that difficulties raised by bribing disappear. Since priority class is assigned on the basis of the bribe, the distribution of bribes is absorbed in the arrival rates. Thus the problem which we have to solve is the single server multiple priority class problem with reneging. An illustration of the model is given in Figure 4.1.

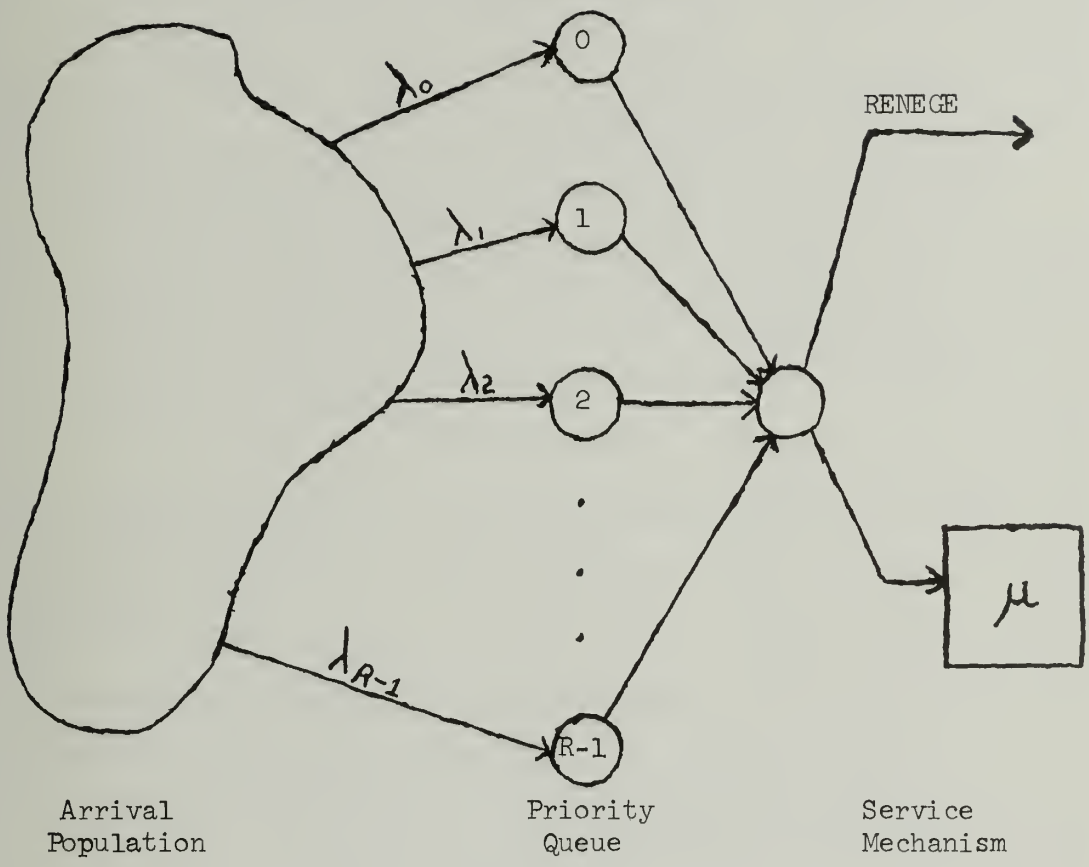


Figure 4.1 A Model of our Queueing System with Reneging

To simplify notation, let us define

$$\sigma_r = \sum_{i=0}^{r} \lambda_i, \quad \sigma = \sigma_{R-1}$$

$$\beta_r = \sum_{i=0}^{r} \alpha_i \quad \beta = \beta_{R-1}.$$

Let $P_{m,r}(t)$ denote the probability that at time t an arrival of priority r finds (m-1) waiting tasks of priority r or higher; that is, he is m^{th} away from service.

As before, let us begin by generating difference equations. Starting with $P_{1,r}(t+\Delta t)$, the probability that at time $t+\Delta t$ a priority r arrival finds himself first in the queue, we note that this probability is given by the sum of the following mutually exclusive and collectively exhaustive events:

a) an arrival of priority r at time t would have been first in the queue and during the interval of length Δt there were no arrivals of priority r or higher (with probability $(1-\sigma_r \Delta t)$), no services (with probability $(1-\mu\Delta t)$, and no reneges from priority r or higher (with probability (1)). This term is given by

$$P_{1,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(1) + o(\Delta t)^2.$$

b) an arrival of priority r at time t would have been second in the queue and during the interval there was no arrival of priority r or higher and either a service or a renege. This term is given by

$$P_{2,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1-\beta_r\Delta t)$$
$$+ P_{2,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(\beta_r\Delta t) + o(\Delta t)^2.$$

c) an arrival of priority r at time t would have found the server idle (and hence gone directly into service) and during the interval there was an arrival of any priority. This term is given by

$$P_0(t)(\sigma\Delta t)(1)(1) + o(\Delta t)^2.$$

d) an arrival of priority r at time t would have been first in the queue and during the interval there was a service. if this occurred and there was more than one customer in the queue, another would have entered service and an arrival at time $t+\Delta t$ would still be first in the queue. This term is given by

$$P_{1,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) - P_1(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) + o(\Delta t)^2.$$

Thus we have

$$\begin{aligned} P_{1,r}(t+\Delta t) = {} & P_{1,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(1) \\ & + P_{2,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1-\beta_r\Delta t) \\ & + P_{2,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(\beta_r\Delta t) \\ & + P_0(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) \\ & + P_{1,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) \\ & - P_1(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) + o(\Delta t)^2. \end{aligned}$$

In a similar manner, we find that for $1 < m \leq M$

$$\begin{aligned} P_{m,r}(t+\Delta t) = {} & P_{m,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(1-(m-1)\beta_r\Delta t) \\ & + P_{m+1,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1-m\beta_r\Delta t) \\ & + P_{m+1,r}(t)(1-\sigma_r\Delta t)(1-\mu\Delta t)(m\beta_r\Delta t) \\ & + P_{m-1,r}(t)(\sigma_r\Delta t)(1-\mu\Delta t)(1-(m-2)\beta_r\Delta t) + o(\Delta t)^2. \end{aligned}$$

These lead to the steady state results

$$P_{2,r} = (\sigma_r/(\mu+\beta_r))\ P_{1,r}$$

$$P_{m+1,r} = ((\sigma_r+\mu+(m-1)\beta_r)/(\mu+m\beta_r))\ P_{m,r} - (\sigma_r/(\mu+m\beta_r))\ P_{m-1,r}$$

for m = 1, 2, ..., M. By induction on m we can show that

$$P_{m+1,r} = ((\sigma_r/\mu)^m/(\prod_{i=1}^{m}(1+i\beta_r/\mu)))\ P_{1,r} \qquad m = 1, 2, \ldots, M.$$

In addition

$$P_{0,r}(t+\Delta t) = P_{0,r}(t)(1-\sigma_r\Delta t)(1)(1) + P_{1,r}(t)(1-\sigma_r\Delta t)(\mu\Delta t)(1) + o(\Delta t)^2$$

so that

$$P_{0,r}'(t) = -\sigma_r P_{0,r}(t) + \mu P_{1,r}(t)$$

or at steady state

$$P_{1,r} = (\sigma_r/\mu)\ P_{0,r}$$

which leads to the result that

$$P_{m+1,r} = ((\sigma_r/\mu)^{m+1}/(\prod_{i=1}^{m}(1+i\beta_r/\mu)))\ P_{0,r} \qquad m = 0, 1, 2, \ldots, M.$$

(We define $\prod_{i=1}^{0}(1+i\beta_r/\mu) = 0.$)

With the steady state probabilities found, it remains to find some representative system descriptors. The one descriptor we are most interested in is the expected waiting time for a priority r task which gives renege time of t. In addition, we find the average number of tasks of priority r or higher and the average number of tasks of

priority r or higher which give renege time less than or equal to t.

Let $W_{r,t}$ denote the waiting time of a priority r task which gives us renege time t. First look at the waiting time for a priority R-1 task. Let n_t be the number of tasks in the queue with renege time less than or equal to t and N be the total number of tasks in the queue. Our tagged task must first of all wait for service to those n_t tasks ahead of him. In addition he must wait for service to those tasks which arrive during his waiting time and take position in the queue ahead of him. Since the expected number of arrivals is $\lambda W_{R-1,t}$ and the probability of an arriving task giving renege time less than or equal to t is $(1-e^{-\alpha t})$ we expect that the number of tasks which will arrive and take position in front of our tagged task is $(1-e^{-\alpha t})\lambda W_{R-1,t}$. Since each unit serviced contributes $1/\mu$ time units to our task's waiting time, the total contribution of these new arrivals plus that of those in the queue already is

$$1/\mu(n_t + (1-e^{-\alpha t})\lambda W_{R-1,t}). \qquad (1)$$

However, during our task's waiting time some of the N tasks already in the queue will renege. By virtue of the ordering of tasks within the queue we know that should a task renege it must be a task positioned ahead of our tagged task in the queue. We expect that, of the N tasks in the queue, $N\alpha W_{R-t,t}$ will renege during our tagged task's waiting time. Thus expression (1) is too large by a factor of $1/\mu N\alpha W_{R-1,t}$. Therefore we immediately find that

$$W_{R-1,t} = 1/\mu(n_t + (1-e^{-\alpha t})\lambda W_{R-1,t}) - 1/\mu N\alpha W_{R-1,t}$$

which reduces to

$$W_{R-1,t} = (n_t/\mu)/(1 + 1/\mu N\alpha - (1-e^{-\alpha t})\lambda/\mu).$$

Let us now examine the case for r = R-2. Let $n_{r,t}$ denote the number of tasks of priority r or higher which give renege time less than or equal to t and let N_r denote the number of tasks of priority r or higher in the queue. Also let β_r and σ_r denote the average renege rate and average arrival rate of tasks of priority r or higher. As before, our task must wait for service to those $n_{r,t}$ tasks already ahead of him in the queue and to those $(1-e^{-\beta_r t})\sigma_r W_{r,t}$ tasks which will arrive and take position in the queue ahead of him. Again, there are $N_r\beta_r W_{r,t}$ tasks which will renege (all of which originate in a queue position ahead of our task) so that

$$W_{r,t} \geq 1/\mu(n_{r,t} + (1-e^{-\beta_r t})\sigma_r W_{r,t}) - 1/\mu N_r\beta_r W_{r,t}$$

which again reduces to

$$W_{r,t} \geq (n_{r,t}/\mu)/(1 + 1/\mu N_r\beta_r - (1-e^{-\beta_r t})\sigma_r/\mu). \qquad (2)$$

Now, however, (2) is really the amount of time which our task would wait if no tasks from a lower priority class were serviced before him. Under our system, however, if there are $1/\mu$ or more time units of float time in priority r and higher, a task of priority r+1 will be serviced. In other words, after tasks from all lower priority classes are scheduled and those of priority r are scheduled there are still $1/\mu$ time units of float time available, then a task of priority r+1 will be processed. Let $P[fl_r \geq 1/\mu]$ denote the probability that the cumulative float time for priority classes r and higher is greater than $1/\mu$. Then $P[fl_r \geq 1/\mu]$ is also the probability that a priority r+1

task is processed if one exists. Since the number of tasks which can be processed during a time interval of length t units is μt, we find that the contribution of priority r+1 (=R-1 in this case) to a priority r task's waiting time is $\mu t P[fl_r \geq 1/\mu] 1/\mu = tP[fl_r \geq 1/\mu]$ since we are effectively conducting μt independent trials with probability of success $P[fl_r \geq 1/\mu]$ each time and contribution $1/\mu$ from each success. Thus we find that (2) is too small by a factor of $tP[fl_r \geq 1/\mu]$ and hence

$$W_{r,t} = \frac{1/\mu n_{r,t}}{1 + 1/\mu N_r \beta_r - (1-e^{-\beta_r t})\sigma_r/\mu} + tP[fl_r \geq 1/\mu]$$

for r = R-2. Continuing this reasoning we find that

$$W_{r,t} = \frac{1/\mu n_{r,t}}{1 + 1/\mu N_r \beta_r - (1-e^{-\beta_r t})\sigma_r/\mu} + \sum_{i=r}^{R-1} tP[fl_i \geq 1/\mu] \qquad (3)$$

for r = 0, 1, ..., R-1 where we define $P[fl_r \geq 1/\mu] = 0$ for r = R-1, to make things more compact.

The average number of tasks of priority r or higher in the queue, N_r, is given by the well known formula

$$N_r = \sum_{m=0}^{M} m p_{m+1,r}$$

where M is the maximum queue size. Substituting earlier results we find that

$$N_r = \sum_{m=0}^{M} m \frac{(\sigma_r/\mu)^m}{\prod_{i=1}^{m} (1+i\beta_r/\mu)} P_{1,r}. \qquad (4)$$

Unfortunately, a closed form of this result is not available.

The more important descriptor is $n_{r,t}$, the average number of tasks of priority r or higher which give renege time less than or equal to t. We find that

$$n_{r,t} = (1-e^{-\beta_r t})\, P_{1,r} \sum_{m=0}^{M} \frac{m(\sigma_r/\mu)^m}{\prod_{i=1}^{m} (1 + i\beta_r/\mu)} \qquad (5)$$

which, even more unfortunately, is not in a closed form.

Substituting equations (4), and (5) into equation (3) completes the development of $W_{r,t}$. These last two descriptors, while useful in their own right, do not make $W_{r,t}$ any simpler. We will content ourselves, therefore, with the expressions in their more compact forms.

We have completed our analysis of priority assignment. Opportunities remain for future research, especially in the queueing theory area. The K processor case is an obvious choice for immediate work. In addition, we expect that our algorithms will be modified as the realities of resource sharing become better known.

5. CONCLUSION

In this paper we have taken a slightly different approach to the study of network computers. Instead of looking at the technology of network computers, we have concerned ourselves with possible methods of making (dare we say that crass word) money. We feel that this approach will find greater popularity as time goes by. Network computers must be treated in the same fashion as any other commodity. This is not a surprising point of view. In the early days of the digital computer, there was widespread opinion that these machines were too expensive and too limited to be of much value to anyone but scientists and census takers. The situation has changed. Computer manufacturers are subject to the same economic principles and constraints as most industries. Given time, network computers will also enter the business world where good management and cost effectiveness reign as king and queen.

We have seen, in Chapter 1, that all network computers have many things in common. The key to cost efficiency is clearly effective resource sharing since, without resource sharing, network computers have nothing of value to offer. However, in Chapter 2, we saw that there is no formula for success in the construction of network computers. The available options are numerous and it is quite possible that networking will be the focus of innovative businessmen for some time to come.

There remain many problems which have not been solved. Many of these are business problems. Perhaps the most difficult of these will be developing a realistic guarantee which will be firm enough to attract contract customers. These customers certainly make up a customer base which must be courted. They are, however, the most difficult customers to attract. They can be expected to demand much and forgive few mistakes. But to the network

which can attract and keep these customers will belong a lucrative and stable base of operations.

Other difficulties also remain. Questions of protocol are difficult and presently poorly answered. Technology still has some work to do in terms of getting prices down and speed and reliability up. Studies which involve using microprogrammable minicomputers for communication purposes appear to be promising. They indicate a forthcoming advance in this area.

The principle roadblock removed by this paper was across the road which leads to successful sharing of common resources. Network computers open up new possibilities for task scheduling. We feel that the algorithms and ideas presented in Chapter 3 represent a first step towards successful customer oriented computing.

In the interest of completeness, we took a stab, in Chapter 4, at solving the previously unconsidered queueing theory problem of priority classes with bribing and reneging. This effort represents an attempt to advance queueing theory in terms of more complex, yet more practical, priorit assignment techniques. The results are not complete, but we are encouraged by the results obtained thus far.

LIST OF REFERENCES

[1] Abramson, Norman, "THE ALØHA SYSTEM," ALØHA SYSTEM Technical Report B72-1, January/1972, 30 pages.

[2] Abramson, Norman, "THE ALØHA SYSTEM--Another Alternative for Computer Communications," ALØHA SYSTEM Technical Report B70-1, ABRIO, 1970, 29 pages.

[3] Adiri, Igal, "A Dynamic Time-Sharing Priority Queue," Journal of the ACM, Vol. 18, No. 4, October, 1971, pp. 603-610.

[4] Adiri, Igal, "A Note on Some Mathematical Models of Time-Sharing Systems," Journal of the ACM, Vol. 18, No. 4, October, 1971, pp. 611-615.

[5] Ancker, C. J. Jr., and Grafarian, A. V., "Queuing with Impatient Customers Who Leave at Random," Journal of Industrial Engineering, Vol. 13, No. 2, March-April, 1962, pp. 84-90.

[6] Ancker, C. J., Jr., and Grafarian, A. V., "Queuing with Reneging and Multiple Heterogeneous Servers," Naval Research Logistics Quarterly, Vol. 10, No. 2, June, 1963, pp. 125-149.

[7] Benes, V. E., "On Some Proposed Models for Traffic in Connecting Networks," The Bell System Technical Journal, Vol. 46, January, 1967, pp. 105-116.

[8] Bortels, W. H., "Simulation of Interference of Packets in time ALØHA Time-Sharing System," ALØHA SYSTEM Technical Report B70-2, March, 1970, 26 pages.

[9] Bowdon, E. K., Sr., "Design Automation in Network Computers," The Ninth Design Automation Workshop Proceedings, 1972, pp. 350-356.

[10] Bowdon, E. K., Sr., "Dispatching in Network Computers," Proc. of the Symposium on Computers-Communications Networks and Teletraffic, April, 1972.

[11] Bowdon, E. K., Sr., "Modeling and Analysis of a Network of Computers," Ph.D. dissertation, University of Iowa, Iowa City, Iowa, 1969.

[12] Bowdon, E. K., Sr., "Network Computer Analysis," Proc. of the Fifth International Conference on System Sciences, January, 1972, pp. 201-203.

[13] Bowdon, E. K., Sr., and Barr, W. J., "Cost Effective Priority Assignment in Network Computers," University of Illinois Department of Computer Science Report No. UIVCDCS-R72-509, March 1972, 23 pages.

[14] Bowdon, E. K., Sr., and Barr, W. J., "Throughput Optimization in Network Computers," Proc. of the Fifth Hawaii International Conference on System Sciences, January, 1972, pp. 528-530.

[15] Carr, C. S., Crocker, S. D., and Cerf, V. G., "Host-Host Communication Protocol in the ARPA Network," Proc. of the Spring Joint Comp. Conf., Vol. 36, 1970, pp. 589-597.

[16] Cobham, A., "Priority Assignments in Waiting Line Problems," Operations Research, Vol. 2, 1954, pp. 70-76, "A Correction," Operations Research, Vol. 3, 1955, p. 547.

[17] Cole, G. P., "Computer Network Measurements: Techniques and Experiments," University of California, Los Angeles School of Engineering and Applied Science. Report No. UCLA-ENG-7165, October, 1971, 350 pages.

[18] Conway, R. W., Maxwell, W. L., and Miller, L. W., Theory of Scheduling, Addison-Wesley Publishing Company, Reading, Mass., 1967.

[19] Farber, D. J., "Networks: An Introduction," Datamation, Vol. 18, No. 4, April, 1972, pp. 36-39.

[20] Farber, D. J., "Progress Report on the Distributed Computing System," University of California at Irvine, Technical Report, January, 1972.

[21] Farber, D. J., and Larson, K. C., Supplement to Proposal for Research Submitted to the National Science Foundation on Distributed Computing System, University of California, Irvine, Technical Report, October, 1970

[22] Farber, D. J., and Larson, K. C., "The System Architecture of the Distributed Computer System--An Informal Description," University of California, Irvine, Technical Report No. 11, Septermber, 1971, 36 pages.

[23] Frank, H., and Frisch, I. T., Communication, Transmission and Transportation Networks, Addison-Wesley Publishing Company, Reading, Massachusetts, 1971.

[24] Frank, H. et. al., "Topological Considerations in the Design of the ARPA Net," Proc. of the Spring Joint Comp. Conf., Vol. 36, 1970, pp. 581-587.

[25] Gilbert, E. N., "Minimum Cost Communication Networks," The Bell System Technical Journal, Vol. 46, November, 1967, pp. 2209-2227.

[26] Greenberger, M., "The Priority Problem and Computer Time Sharing," Management Science, Vol. 12, No. 11, July, 1966, pp. 888-906.

[27] Gupta, S. F., and Goyal, J. K., "Queues with Poisson Input and Hyper-Exponential Output with Finite Waiting Space," Operations Research, Vol. 12, No. 1, 1964, pp. 75-81.

[28] Heart, F. E., et. al., "The Interface Message Processor for the ARPA Computer Network," Proc. of the Spring Joint Comp. Conf., Vol. 36, 1970, pp. 551-567.

[29] Hootman, J. T., "The Computer Network as a Marketplace," Datamation, Vol. 18, No. 4, April, 1972, pp. 43-46.

[30] Jackson, R. R. P., and Henderson, J. C., "The Time-Dependent Solution to the Many-Server Poisson Queue," Operations Research, Vol. 14, 1966, pp. 720-722.

[31] Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," Proc. of the Spring Joint Comp. Conf., Vol. 36, 1970, pp. 569-579.

[32] Kleinrock, L., Communication Nets: Stochastic Message Flow and Delay, McGraw-Hill, New York, 1964.

[33] Kleinrock, L., "Optimum Bribing for Queue Position," Operations Research, Vol. 15, 1967, pp. 304-318.

[34] Kruskal, J. B., "Work-Scheduling Algorithms: A Nonprobabilistic Queuing Study (With Possible Application to No. 1 ESS)," The Bell System Technical Journal, Vol. 48, November, 1969, pp. 2963-2974.

[35] Mosmann, C., and Stefferud, E., "Computer Management," College and University Business, January, 1971, pp. 50-52.

[36] Mossman, C., and Stefferud, E., "Campus Computing Management," Datamation, Vol. 17, No. 5, March, 1971.

[37] Parker, L. T., Jr., Gallie, T. M., Brooks, F. P., Jr., and Ferrell, J. K., "Introducing Computing to Smaller Colleges and Universities--a Progress Report," Comm. ACM, Vol. 12, No. 6, June, 1969, pp. 319-323.

[38] Roberts, L. G. and Wessler, B. D., "Computer Network Development to Achieve Resource Sharing," Proc. of Spring Joint Comp. Conf., Vol. 36, 1970, pp. 543-549.

[39] Saaty, T. l., Elements of Queueing Theory, McGraw-Hill, New York, 1961.

[40] Stefferud, E., "Management's Role in Networking," Datamation, Vol. 18, No. 4, April, 1972, pp. 40-42.

[41] Stefferud, E., "The Environment of Computer Operating System Scheduling: Toward an Understanding," AEDS Journal, March, 1968, pp. 135-141.

[42] Syski, R., Introduction to Congestion Theory in Telephone Systems, Oliver and Boyd, Edinburgh, 1960.

[43] Williams, L. H., "A Functioning Computer Network for Higher Education in North Carolina," Submitted for Fall Joint Computer Conference, 1972.

[44] Private conversation with Michael Sher, Center for Advanced Computation, University of Illinois, Urbana, Illinois.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-72-538	2.	3. Recipient's Accession No.

4. Title and Subtitle

Cost Effective Analysis of Network Computers

5. Report Date

August 1972

6.

7. Author(s)

William J. Barr

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No.

NSF GJ 28289

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D.C.

13. Type of Report & Period Covered

Thesis Research

14.

15. Supplementary Notes

16. Abstracts

With the advent of network computers, a new area of computer systems analysis has evolved. Unfortunately, most of the work to date merely extends the previously existing theory of communications. Our analysis is predicated on the assumption that a geographically distributed network computer is quite different from telephone networks and individual computing centers. The potential advantages, from a businessman's point of view, of network computers are discussed and some of the more important problems which arise are presented. We take a look at several existing networks and examine their goals and solutions to these kinds of problems. We develop a priority assignment technique for the individual centers that comprise the network and determine load leveling rules for the entire network. In addition, we analyze this sytem using classical queueing theory techniques.

17. Key Words and Document Analysis. 17a. Descriptors

Network Computer, Queueing Theory, System Modeling, Throughput Analysis, Dispatching, Load Leveling

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Unlimited Distribution	UNCLASSIFIED	73
	20. Security Class (This Page) UNCLASSIFIED	22. Price None